

IEEE Standard for Futurebus+ — Physical Layer and Profile Specification

Sponsor
**Bus Architecture Standards Committee
of the
IEEE Computer Society**

Approved September 26, 1991
IEEE Standards Board

Abstract: Futurebus+ standards provide systems developers a set of tools with which high-performance bus-based systems may be developed. This architecture provides a wide range of performance scalability over both cost and time for multiple generations of single- and multiple-bus multiprocessor systems. This document, a companion standard to IEEE Std 896.1-1991, builds on the logical layer by adding requirements for physical layer instantiation. Material in this document includes specifications for node management, live insertion, and profiles. It is to these profiles that products will claim conformance. Other specifications that may be required in conjunction with this standard are the following: IEEE Std 896.1-1991; P896.3, Futurebus+ Recommended Practices; P1212.x, Control and Status Register Architectures; IEEE Std 1194.1-1991, Electrical Characteristics of Backplane Transceiver Logic (BTL) Interface Circuits; and IEEE Std 1301.x, Metric Equipment Practices for Microcomputers.

Keywords: application environment profiles, bus architecture, Futurebus+, live insertion, multiprocessor systems, node management, physical layer

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

© 1992 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published in 1992
Printed in the United States of America

1-55937-169-2

*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 Technical Committees of 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 which 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 once 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 technical 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

IEEE Standards documents are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Foreword

(This foreword is not a part of IEEE Std 896.2-1991, IEEE Standard for Futurebus+ — Physical Layer and Profile Specification.)

Historical Perspective

IEEE Std 896.2-1991 defines the first set of profiles for the IEEE Futurebus+ family of standards. The IEEE Futurebus+ Working Group voted to develop a separate specification to contain the physical layer and profiles for the Futurebus+ standard in the July 1989 Meeting of the Working Group in Milpitas, California. The Project Application Request (PAR) was approved by the IEEE Microprocessor Standards Committee for submission to the IEEE Standards Board in September 1989, and the project was approved with the number P896.2 by the IEEE Standards Board in November 1989.

Purpose and Features of the Futurebus+ Specifications

A unique attribute of the design of Futurebus+ protocols is the protocols' technology independence, achieved through basing them on fundamental protocol and physics principles and optimizing them for maximum communication efficiency (and, hence, throughput), rather than for a particular generation or type of processor. Timing and handshake protocols are governed by "law of nature" types of constraints rather than limitations of current and projected technology, such as skews, device propagation delays, and capture windows.

The benefits of technology independence are apparent in the principle of no technology-based upper limit to the performance of Futurebus+. Configuration and transaction capability modes guarantee interoperability when two devices of different speeds or of different generations communicate on the same bus segment, thus providing Futurebus+ with an unprecedented ability to support multiple generations of computers well into the twenty-first century.

Because of this, Futurebus+ is layered into a family of standards, which separates the fundamental architectural framework of the standard, and the logical protocols which serve that architecture, from the physical implementation of those protocols in a packaging scheme suitable for the intended application.

IEEE Std 896.1-1991 protocols may therefore be implemented using *any* logic family (e.g., TTL, BTL, CMOS, ECL, GaAs) providing the physical implementation is constrained such that it meets the Futurebus+ signaling requirements (incident wave switching and positive relative skews between information and synchronization lines).

Additionally, the protocols in IEEE Std 896.1-1991 may be used at *any* level in the system hierarchy: chip-to-chip, board-to-board, system-to-system, or at several levels at once in the same system. These protocols are particularly powerful when used at two or more levels in a hierarchical bus system; this was an intended aspect of the protocol design, since today's backplane protocols often become tomorrow's chip-to-chip protocols.

Profiles and Interoperability

This flexibility of logical and physical implementation of the Futurebus+ architecture presents a dilemma: how to minimize unnecessary variations in products developed by many different vendors. The problem is addressed by the use of *profiles* in this document. Profiles tie together the various specifications and support standards to create interoperability. Restrictions in profile specifications are designed to ensure compatibility amongst products. The word *conformance*, therefore has no meaning within the IEEE Futurebus+ family of specifications, except when used in conjunction with an approved profile.

This document contains the first three profiles developed by the working group. Each has demonstrated a real "need to exist" based on major differences in the intended application.

Profile A:	General Purpose Multiprocessor Systems
Profile B:	General Purpose I/O Systems
Profile F:	Very High Performance Multiprocessor Systems

The working group, recognizing the diverse compute environs served by the Futurebus+ protocols, are working on the following profiles for future release:

Profile T:	Telecommunications
Profile M:	Military Systems
Profile D:	Desktop Systems
Profile C:	Futurebus+ Cable

Specifications that may be treated as a family for use within a typical Futurebus+ profile are IEEE Std 896.1-1991, IEEE Standard for Futurebus+ — Logical Protocol Specification; IEEE Std 896.2-1991, IEEE Standard for Futurebus+ — Physical Layer and Profile Specification; IEEE Std 1194.1-1991, IEEE Standard for Electrical Characteristics of Backplane Transceiver Logic (BTL) Interface Circuits; IEEE Std 1212-1991, IEEE Standard Control and Status Register (CSR) Architecture for Microcomputer Buses; IEEE Std 1301-1991 and IEEE Std 1301.1-1991 (metric mechanical standards); and P896.3, Futurebus+ Recommended Practices.

A profile consists of a set of choices of specification sections, and options within those sections, that are appropriate for use together within an implementation. The explicit discussion of what is required and what is not, within a given profile, addresses the interoperability issues brought about by arbitrary assignment of optional features. Profiles also allow the buyer of Futurebus+-based products to know exactly what features come with each product. If a manufacturer follows the requirements laid out within a profile, that product is much more likely to be interoperable with the products of any other manufacturer meeting that profile.

Though profiles address different applications, and interoperability across multiple profiles cannot be guaranteed, there are a number of advantages to using the same basic architectural foundation and protocols:

- They can share silicon support; for example, arbitration and parallel protocol controllers, memory controllers, cache controllers, serial bus controllers, and diagnostic interface devices.
- The learning experience of engineers implementing an interface for one product is directly applicable to the next (possibly different) product, thus shortening product development cycles.
- Two or more products using different profiles may intercommunicate using an interface cable between those systems, and maintain architectural continuity throughout the larger interconnected system. This is particularly useful for allowing, for example, desktop computers using Futurebus+ to be part of the same shared-memory environment as a large powerful multiprocessor array. The desktop bus may use CMOS, while the large machine may use ECL, but the architecture is maintained and efficiency is not lost due to converting from one protocol to another.
- An infrastructure can be created within the silicon support and software architecture so that diagnostics (JTAG support), configuration, initialization, and device driver interfaces can be standardized.

Futurebus+ sets the scene for a major paradigm shift within the computer industry. No longer is it necessary to think of isolated islands of computing power having to be interconnected with grossly inefficient intermediate protocols in order to overcome the tower of Babel created by the parochial interests of competing vendors and the unyielding march of technology obsoleting over-optimized products before they leave the CAD screen. The stage has been set for systems to *conveniently* communicate with one another; all can now speak the same language, regardless of the application for which they were designed: Futurebus+.

Acknowledgements

The Futurebus+ project has attracted some of the most capable and dedicated engineers and computer scientists in the industry to help develop the specifications. This is true not

only for the architecture and protocols specified in IEEE Std 896.1-1991, but continues to be true for the electrical and mechanical specifications and all the operational requirements specified in this document. Those individuals who contributed most to the development of this document are as follows:

Barbara Aichinger	Digital Equipment Corp.
John Allen	Data General
Jeff Beachy	Tektronix, Inc.
Harrison Beasley	Texas Instruments
Mark Bunker	Raytheon
Karen Churas	Raytheon
Sam Duncan	Digital Equipment Corp.
Wayne Fischer	Force Computers
Joe George	Nanotek
Claes Goren-Gustavsson	ABB
Bob Greiner	Motorola
Emil Hahn	Signetics
George Harris	Digital Equipment Corp.
Ed Jacques	John Hopkins U. APL
David V. James	Apple Computer, Inc.
Dick Lawrence	Digital Equipment Corp.
Jim Leahy	Digital Equipment Corp.
Dave Moore	Digital Equipment Corp.
Brian Morrison	Raytheon
Chris Nichols	Ironics, Inc.
William Samaras	Digital Equipment Corp.
Don Senzig	Amdahl Corp.
John Theus	The Theus Group
Patricia Smith	Naval Avionics Center
Eike Waltz	Schroff, Inc.
David Wright	896 Computers, Ltd.

A special thanks for their exceptional hard work is made to Harrison Beasley of Texas Instruments, Vice Chairman of the P896.2 Project, and to Thanos Mentzelopoulos of Ironics, Inc., and Ernie Crocker of Digital Equipment Corp., draft editors for the project.

The following is a list of those who were members of the IEEE Futurebus+ Working Group at the time this standard was approved:

Paul L. Borrill, Chair

Barbara Aichinger	Joseph George	Clarence Peckham
Ray Alderman	Larry Gilbert	Shlomo Pri-Tal
Hamid Amirazzi	Jim Goodman	Surinder Rai
Duane Anderson	Robert Greiner	Mike Raynham
Harrison Beasley	David Gustavson	Jack Regula
Janos Biri	David Hartig	Bill Ruszczyk
Martin Blake	Emil Hahn	Ali Sarabi
Richard Boberg	David Hawley	James Scaminaci
Andy Bonaffini	Lym Hevle	Dennis Schmitz
David Brash	Billy Ho	Craig Scott
David Brearly	Mike Humphrey	Don Senzig
David Brewer	John Hyde	Dan Sieworek
Marc Briel	Ed Jacques	Lui Sha
Charles Brill	David James	Mike Snodgrass
Jim Brown	Greg Jewell	Michael Sweeney
Mark Bunker	Anatol Kaganovich	Fahad Tabrizzi
John Campbell	Hans Karlsson	Matthew Taub
Jay Cantrell	David Kemp	Mike Teener
Stephen Cecil	Ralph Lachenmaier	Judy Teske
Kim Clohessy	Subasis Laha	Morton Thayer
Paul Cook	Cees Lambretche	Mike Thompson
Dante Del Corso	Dick Lawrence	Nigel Topham
Ernie Crocker	Mike Lazar	John Theus
Jon Crowell	Jim Leahy	Mary Vernon
Steve Diess	Kent Leung	Harvey Walthersdorf
Paul Dixon	Joel Liblove	Eike Waltz

Ian Dobson
Emer Dooley
Sam Duncan
Chris Eck
Bill Evertz
Wayne Fischer
Mike Foster

Thanos Mentzelopoulos
Klaus-Dieter Mueller
Chris Nichols
Jim Nicholson
Ronald Niederhagen
Mira Pauker
Chet Pawlowski

Randy Weber
Mike Wenzel
Mike Wiles
Mark Williams
John Wise
David Wright
Dale Younge

The following persons were on the balloting committee that approved this document for submission to the IEEE Standards Board:

M. R. Aaron
William B. Adams
Barbara P. Aichinger
Ray S. Alderman
Richard P. Ames
Duane L. Anderson
Peter J. Ashenden
Harrison A. Beasley
Russell A. Beverly
Janos Biri
John C. Brightwell
Charles Brill
Myron Calhoun
Kim Clohessy
Rob Colclaser
Steve Cooper
Stephen Deiss
Dante Del Corso
Jean-Jacque Dumont
Samuel Duncan
William P. Evertz
Wayne Fischer
Gordon Force
Andrew Fraser
Tayeb Giuma
Andy Glew
Jack Goldstein
Patrick Gonia

David B. Gustavson
Emil Hahn
Mark Hassel
David Hawley
Michael C. Hayward
Frank Hom
Edgar Jacques
Jack R. Johnson
Hans Karlsson
Christopher Koehle
Ernst H. Kristiansen
Ralph Lachenmaier
Gerry Laws
Michael E. Lee
Thomas Leonard
Rollins Linser
William McDonald
James M. Moidel
Brian D. Morrison
Klaus Dieter Mueller
Cuong Nguyen
J. D. Nicoud
Daniel C. O'Connor
Richard W. Palmer
Francisco Pataro
Mira Pauker
Donald Pavlovich
Shlomo Pri-Tal

Richard Rawson
Ray Steven
Paul Rosenberg
James Seago
Don Senzig
Philip Shutt
Patricia Smith
David Stevensen
Daniel Tabak
Ahmed N. Tantawy
Daniel Tarrant
Michael Teener
Joseph P. Trainor
Robert Tripi
Robert J. Voigt
Yoshiaki Wakimura
Eike Waltz
Carl Warren
Dave Wickliff
Robert Widlicka
Dwight Wilcox
John S. Willy
Andrew Wilson
John Wise
Joel Witt
David L. Wright
Oren Yuen
Janusz Zalewski

When the IEEE Standards Board approved this standard on September 26, 1991, it had the following membership:

Marco W. Migliaro, Chair

Donald C. Loughry, Vice Chair

Andrew G. Salem, Secretary

Dennis Bodson
Paul L. Borrill
Clyde Camp
James M. Daly
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ingrid Fromm

Thomas L. Hannan
Donald N. Heirman
Kenneth D. Hendrix
John W. Horch
Ben C. Johnson
Ivor N. Knight
Joseph L. Koepfinger*
Irving Kolodny
Michael A. Lawler

John E. May, Jr.
Lawrence V. McCall
Donald T. Michael*
Stig L. Nilsson
John L. Rankine
Ronald H. Reimer
Gary S. Robinson
Terrance R. Whittemore

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Fernando Aldana
Satish K. Aggarwal
James Beall
Richard B. Engelman
Stanley Warshaw

Paula M. Kelty
IEEE Standards Project Editor

Contents

SECTION	PAGE
1. Introduction.....	17
1.1 Scope.....	17
1.2 References.....	18
1.2.1 Protocol Stack Specifications.....	18
1.2.2 International Government Regulations for Safety and EMI/RFI Control.....	18
2. Definitions and Structure.....	19
2.1 Special Word Usage.....	19
2.2 Definitions.....	20
2.3 Signal Conventions.....	24
2.4 Numbering Conventions.....	24
2.5 Futurebus+ Logo.....	25
3. Node Management.....	26
3.1 Introduction.....	26
3.1.1 Address Space.....	26
3.1.1.1 Memory Addresses.....	26
3.1.1.2 CSR Addresses.....	26
3.1.1.3 Hybrid Addressing.....	28
3.1.1.4 Byte Lane Mapping.....	29
3.1.2 Base Functional Character.....	30
3.1.2.1 Resets.....	31
3.1.2.2 ROM Identification.....	31
3.1.2.3 Capability and Setup.....	32
3.1.2.4 Bus Control.....	32
3.1.2.5 Distributed Arbitration.....	33
3.1.2.6 Timeout and Setup.....	33
3.1.2.7 Retry.....	34
3.1.2.8 Node State Control.....	34
3.1.2.9 Standard Diagnostic Interface.....	35
3.1.3 Optional Functional Character.....	37
3.1.3.1 Messages.....	37
3.1.3.2 Arbitrated Messages.....	38
3.1.3.3 Interrupts.....	39
3.1.3.4 Packet Management.....	40
3.1.3.5 Clock Synchronization.....	40
3.1.3.6 Unit Access.....	44
3.2 Futurebus+ CSR Specification.....	46
3.2.1 CSR Memory Space Specification.....	46
3.2.2 Core CSRs.....	47
3.2.2.1 STATE_CLEAR and STATE_SET CSRs.....	48
3.2.2.2 NODE_IDS CSR.....	50
3.2.2.3 RESET_START CSR.....	51
3.2.2.4 Internal Indirect Address Space CSRs.....	52
3.2.2.5 SPLIT_TIMEOUT CSR.....	53
3.2.2.6 ARGUMENT CSRs.....	54
3.2.2.7 TEST_START CSR.....	55
3.2.2.8 TEST_STATUS CSR.....	58
3.2.2.9 Extended Units Address Space CSRs.....	60
3.2.2.10 Memory Unit Address Space CSRs.....	62

SECTION	PAGE
3.2.2.11	INTERRUPT_TARGET and INTERRUPT_MASK CSRs.....63
3.2.2.12	CSRs for Clock Synchronization 64
3.2.2.13	Message Target CSRs.....67
3.2.2.14	Error Symptom CSRs.....67
3.2.3	Futurebus+ Dependent CSR Area.....74
3.2.3.1	LOGICAL_COMMON_CONTROL CSR.....75
3.2.3.2	LOGICAL_MODULE_CONTROL CSR.....77
3.2.3.3	BUS_PROPAGATION_DELAY CSR.....79
3.2.3.4	COMPETITION_SETTLING_TIME CSR.....80
3.2.3.5	TRANSACTION_TIMEOUT CSR.....80
3.2.3.6	MESSAGE_PASSING_SELECT_MASK CSR.....81
3.2.3.7	Busy Retry CSRs.....81
3.2.3.8	Error Retry CSRs.....83
3.2.4	Bus Specific ROM Entries.....85
3.2.4.1	Bus Information Block.....87
3.2.4.2	MODULE_LOGICAL_CAPABILITY.....87
3.2.4.3	NODE_CAPABILITIES_EXT.....91
3.2.4.4	COMPETITION_INTERNAL_DELAY CSR.....92
3.2.4.5	PACKET_SPEED CSR.....93
3.2.4.6	MESSAGE_FRAME_SIZE CSR.....93
3.2.4.7	Busy Retry Capability CSRs.....94
3.2.4.8	Error Retry Capability CSRs.....94
3.2.5	Core CSR Root Directory.....95
3.2.5.1	MODULE_VENDOR_ID ROM Entry.....97
3.2.5.2	MODULE_SPEC_ID ROM Entry.....97
3.2.5.3	NODE_CAPABILITIES ROM Entry.....97
3.2.5.4	MODULE_HW_VERSION ROM Entry.....99
3.2.5.5	Bus Visible Address RAM.....99
3.2.5.6	Vendor-Defined ROM Entries.....101
3.2.5.7	MODULE_VENDOR_DEPENDENT_INFO ROM Entry.....101
3.2.6	Initial Units Space.....101
3.2.7	Unit-Specific Interrupts.....101
3.2.8	Distributed Arbitration Messages and General Arbitrated Messages.....102
4.	Live Insertion.....103
4.1	Description.....103
4.1.1	Introduction.....103
4.1.2	General Considerations.....103
4.1.3	Levels of Live Insertion.....103
4.1.3.1	Level 1 Live Insertion and Withdrawal.....104
4.1.3.2	Level 2 Live Insertion and Withdrawal.....104
4.1.3.3	Level 3 Live Insertion and Withdrawal.....104
4.1.4	Operator Facilities.....105
4.1.5	Live Insertion State Diagram.....105
4.1.6	Conceptual Module Design.....107
4.2	Specification.....109
4.2.1	Definitions.....109
4.2.2	System Operation Requirements.....109
4.2.3	Live Insertion Power Requirements.....110
4.2.4	Module Activation and Deactivation.....110
4.2.5	Live Insertion ESD Requirements.....110
4.2.6	Live Insertion Electrical Requirements.....110
4.2.6.1	Level 1 Requirements.....110

SECTION	PAGE
4.2.6.2	Level 2 Requirements..... 110
4.2.6.3	Level 3 Requirements..... 111
4.2.7	Futurebus+ Live Insertion Glyph..... 111
4.2.8	Definition of SWAP Indicator..... 111
5.	Introduction to Application Environment Profiles..... 112
5.1	Application Environment Profile (AEP) Description..... 112
5.2	Minimum Requirements for AEPs (AEP Specification)..... 113
5.2.1	AEP Terminology and Definitions..... 113
5.2.2	AEP Organization..... 113
5.2.3	Profile Content..... 114
5.2.3.1	Arbitration..... 114
5.2.3.2	Parallel Protocol..... 114
5.2.3.3	Timing Specification..... 115
5.2.3.4	Bus Management and CSRs..... 116
5.2.3.5	Cache Coherence..... 116
5.2.3.6	Message Passing..... 116
5.2.3.7	Systems Configurations..... 116
5.2.4	Physical Layer..... 116
5.2.4.1	Power Supply Specifications..... 116
5.2.4.2	Profile Electrical and Signaling Environment..... 117
5.2.4.3	Live Insertion..... 118
5.2.4.4	Mechanical..... 118
5.2.4.5	Input/Output..... 119
5.2.4.6	Connector and Pinout Assignment..... 119
5.2.5	Environmental..... 119
5.2.6	Standards Requirements..... 119
6.	Application Environment Profile A..... 120
6.1	Reference Specification..... 120
6.1.1	Introduction..... 120
6.1.2	Target Market and Applications..... 120
6.1.3	Terminology..... 120
6.1.4	Referenced Documents..... 120
6.1.5	Reference Tables..... 120
6.1.6	Profiles Interoperability..... 120
6.2	Detailed Specification..... 125
6.2.1	Arbitration..... 125
6.2.2	Parallel Protocol..... 126
6.2.2.1	Transaction Types..... 126
6.2.2.2	Tag Bits..... 127
6.2.2.3	Serial Bus..... 127
6.2.3	Bus/Node Management and CSRs..... 127
6.2.3.1	Addressing..... 127
6.2.3.2	Byte-Lane Wiring and Byte Ordering..... 127
6.2.3.3	Interrupts..... 127
6.2.3.4	Diagnostics and Test..... 127
6.2.3.5	CSRs..... 128
6.2.4	Cache Coherence..... 131
6.2.5	Message Passing..... 131
6.2.6	System Configuration..... 131
6.2.6.1	Monarch Selection..... 132
6.2.7	Profile A Power..... 132

SECTION	PAGE
6.2.7.1	Module Power 132
6.2.7.2	System Power Supplies..... 132
6.2.8	Profile A Electrical..... 134
6.2.8.1	Signal Integrity..... 134
6.2.8.2	Functional Electrical Requirements 135
6.2.9	Live Insertion and Withdrawal..... 135
6.2.9.1	Alignment..... 136
6.2.9.2	Live Insertion Power..... 136
6.2.9.3	Live Insertion and Withdrawal Safety..... 136
6.2.10	Mechanical..... 137
6.2.10.1	Subrack Mechanical Specifications..... 137
6.2.10.2	Plug-In Unit (Module) Dimensions, Tolerances, and Design Features 144
6.2.10.3	EMI Considerations..... 150
6.2.10.4	Injector/Ejector Mechanism..... 152
6.2.10.5	Bus Connector Mechanical..... 153
6.2.11	Input/Output..... 153
6.2.12	Profile A Connector, Power, and Signal Pin Assignment 153
6.2.12.1	Connector Naming Conventions and Connector Assignment..... 153
6.2.12.2	Signal Pin Assignment..... 154
6.2.12.3	Power Pin Assignment..... 155
6.2.12.4	Connector Keying..... 155
6.2.13	Environmental Specifications and Other Standards Compliance... 155
6.2.13.1	Subrack Air Flow and Thermal Considerations:..... 160
6.2.13.2	Safety and EMI Standards..... 161
7.	Application Environment Profile B 162
7.1	Reference Specification..... 162
7.1.1	Introduction..... 162
7.1.2	Target Market and Applications 162
7.1.3	Profile B Terminology..... 162
7.1.4	Referenced Documents 162
7.1.5	Reference Tables..... 162
7.1.6	Multiple Profile Interoperability 166
7.2	Detailed Specification..... 166
7.2.1	Arbitration 166
7.2.1.1	Central Arbitration..... 166
7.2.1.2	Distributed Arbitration 167
7.2.2	Parallel Protocol..... 167
7.2.2.1	Transaction Types..... 167
7.2.2.2	Prohibited Operations..... 168
7.2.2.3	Timing Specifications..... 169
7.2.2.4	Tag Bits..... 170
7.2.2.5	Serial Bus..... 171
7.2.3	Bus/Node Management and CSRs 171
7.2.3.1	Addressing..... 171
7.2.3.2	Byte-Lane Wiring and Byte Ordering..... 171
7.2.3.3	Interrupts..... 171
7.2.3.4	DMA..... 171
7.2.3.5	Diagnostics and Test..... 171
7.2.3.6	CSRs..... 171
7.2.4	Caching and Cache Coherence..... 174

SECTION	PAGE
7.2.5	Message Passing..... 174
7.2.6	System Configuration..... 175
7.2.7	Profile B Power 175
7.2.7.1	Module Power 175
7.2.7.2	System Power Supplies 175
7.2.8	Profile B Electrical..... 179
7.2.8.1	Signal Integrity..... 179
7.2.8.2	Functional Electrical Requirements 181
7.2.9	Live Insertion and Withdrawal..... 183
7.2.9.1	Level 1 Live Insertion..... 183
7.2.9.2	Level 2 Live Insertion..... 184
7.2.9.3	Alignment..... 184
7.2.9.4	Live Insertion Power..... 184
7.2.9.5	Live Insertion and Withdrawal Safety..... 184
7.2.10	Mechanical..... 184
7.2.10.1	Subrack Mechanical Specifications..... 185
7.2.10.2	Plug-In Unit (Module) Dimensions, Tolerances, and Design Features 187
7.2.10.3	EMI Considerations..... 196
7.2.10.4	Injector/Ejector Mechanism..... 198
7.2.10.5	Bus Connector Mechanical..... 199
7.2.11	Input/Output..... 199
7.2.12	Profile B Connector, Power, and Signal Pin Assignment..... 199
7.2.12.1	Connector Naming Conventions and Connector Assignment..... 199
7.2.12.2	Signal Pin Assignment 200
7.2.12.3	Power Pin Assignment..... 203
7.2.13	Environmental Specifications and Other Standards Compliance... 204
7.2.13.1	Subrack Air Flow and Thermal Considerations..... 204
7.2.13.2	Safety and EMI Standards..... 204
8.	Application Environment Profile F..... 205
8.1	Reference Specifications 205
8.1.1	Introduction..... 205
8.1.2	Target Market and Applications 205
8.1.3	Profile F Terminology..... 205
8.1.4	IEEE Standards Referenced..... 205
8.1.5	Reference Tables..... 205
8.1.6	Multiple Profile Interoperability 207
8.1.6.1	Profile A Interoperability..... 207
8.1.6.2	Profile B Interoperability..... 207
8.2	Detailed Specification 208
8.2.1	Arbitration 208
8.2.2	Parallel Protocol..... 208
8.2.2.1	Transaction Types..... 209
8.2.2.2	Split Transactions..... 209
8.2.2.3	Locked Operations..... 209
8.2.2.4	Packet Data Transfers..... 209
8.2.2.5	Busy..... 210
8.2.2.6	Tag Bits..... 210
8.2.2.7	Serial Bus..... 210
8.2.3	Timing Specifications..... 211
8.2.4	Bus/System Management and CSRs..... 214

SECTION	PAGE
8.2.4.1	Addressing..... 214
8.2.4.2	Byte-Lane Wiring and Byte Ordering..... 214
8.2.4.3	Interrupts..... 214
8.2.4.4	Diagnostics and Test..... 214
8.2.4.5	CSRs..... 214
8.2.5	Caching and Cache Coherence..... 217
8.2.6	Message Passing..... 217
8.2.7	System Configuration..... 217
8.2.7.1	Monarch Selection Description..... 217
8.2.7.2	Monarch Selection Attribute Specification 218
8.2.8	Power Supply..... 218
8.2.9	Electrical..... 218
8.2.9.1	Wire-OR Glitch Filters..... 218
8.2.10	Live Insertion and Withdrawal..... 218
8.2.11	Mechanical..... 218
8.2.12	Input/Output..... 218
8.2.13	Connector, Power, and Signal Pin Assignment..... 218
8.2.13.1	Non-Central Arbiter Slot 218
8.2.13.2	Central Arbiter Slot..... 218
8.2.14	Environmental..... 220
8.3	Specification for a Module-Based Central Arbiter..... 220
8.3.1	Backplane Requirements 220
8.3.1.1	Wiring..... 220
8.3.1.2	Geographical Address Assignment..... 220
8.3.2	Backplane Delay Arbitration Message..... 221
8.3.3	System Reset 221
8.3.4	AC Voltage Low..... 221
8.3.5	Electrical..... 221

FIGURES

Fig 1	Signal Conventions..... 24
Fig 2	Numbering Conventions 25
Fig 3	IEEE 896 Logo 25
Fig 4	Futurebus+ Boards, Modules, and Nodes..... 27
Fig 5 (a)	32-bit CSR Addressing 27
Fig 5 (b)	32-bit CSR Addressing 28
Fig 5 (c)	First CSR Address of Local Bus Node One..... 28
Fig 6 (a)	Native Big Endian Node to Futurebus+ CSR Byte Lane Mapping..... 30
Fig 6 (b)	Native Little Endian Node to Futurebus+ CSR Byte Lane Mapping 30
Fig 7	Clock Synchronization 45
Fig 8	Fields in the STATE_CLEAR and STATE_SET CSRs Used by Futurebus+ ... 50
Fig 9	Format of the NODE_IDS CSR..... 51
Fig 10	Format of the RESET_START CSR..... 52
Fig 11	Format of the Internal INDIRECT_ADDRESS CSR 52
Fig 12	Format of the Internal INDIRECT_DATA CSR..... 53
Fig 13	Format of the SPLIT_TIMEOUT CSRs..... 54
Fig 14 (a)	General Format of the ARGUMENT CSRs..... 55
Fig 14 (b)	Extended Test Category Format of the Argument CSRs..... 55
Fig 15	Format of the TEST_START CSR..... 57
Fig 16	Format of the TEST_STATUS CSR 60
Fig 17	Format of the UNITS_BASE CSR..... 61
Fig 18	Format of the UNITS_BOUND CSR..... 61

FIGURES	PAGE
Fig 19	Format of the MEMORY_BASE CSR.....62
Fig 20	Format of the MEMORY_BOUND CSR.....63
Fig 21	Format of the CLOCK_STROBE CSR 66
Fig 22(a)	ERROR_HI CSR Format (Most Significant Half) 70
Fig 22 (b)	ERROR_HI CSR Format (Least Significant Half).....70
Fig 23	Format of the ERROR_LO CSR.....72
Fig 24	Format of the TAG_HOLD Register.....74
Fig 25	Format of the LOGICAL_COMMON_CONTROL CSR 76
Fig 26	Format of the LOGICAL_MODULE_CONTROL CSR.....79
Fig 27	BUSY_RETRY_COUNTER CSR.....82
Fig 28	BUSY_RETRY_DELAY CSR.....83
Fig 29	ERROR_RETRY_COUNTER CSR.....84
Fig 30	ERROR_RETRY_DELAY CSR.....85
Fig 31 (a)	MODULE_LOGICAL_CAPABILITY (Most Significant Half).....90
Fig 31 (b)	MODULE_LOGICAL_CAPABILITY (Least Significant Half).....90
Fig 32	Format of the NODE_CAPABILITIES_EXT CSR 92
Fig 33	Format of the COMPETITION_INTERNAL_DELAY CSR.....93
Fig 34	Format of the PACKET_SPEED CSR.....93
Fig 35	Format of the MESSAGE_FRAME_SIZE CSR 94
Fig 36	Format of the NODE_CAPABILITIES Entry.....99
Fig 37	Format of the NODE_MEMORY_EXTENT Entry 100
Fig 38	Format of the NODE_UNITS_EXTENT Entry 101
Fig 39	Live Insertion Module State Diagram 106
Fig 40	Functional Design of a Live Insertion Module 109
Fig 41	Futurebus+ Live Insertion Glyph 111
Fig 42	Futurebus+ Open Systems Protocol Stack.....112
Fig 43	Profile A Power Rails.....134
Fig 44	View of Subrack/Backplane Relationship.....138
Fig 45	View of Subrack with Inspection Dimension to the Bottom of the Installed Connector.....139
Fig 46	View of Plug-in Module.....140
Fig 47	Detail “C,” Alignment Pin Position on Front Panel and Subrack 141
Fig 48	Detail “B,” Front Panel Section.....142
Fig 49	Detail “A,” Side View of Front Panel Detail with Injector/Ejector Envelope and Alignment Pin Position.....143
Fig 50	Intermodule Separation Planes, Clearances, and Maximum Component Heights, X-Z View 146
Fig 51	Profile A Front Panel Organization 148
Fig 52	Module Status State Diagram 150
Fig 53	EMI/RFI Shielding Locations and Interfacing Surfaces—Preferred 151
Fig 54	EMI/RFI Shielding Locations and Interfacing Surfaces Alternative 152
Fig 55	Connector Naming Conventions 154
Fig 56	Connector B and X Pinouts for 64/32-Bit Address and Data Paths 156
Fig 57	Connector E Pinout for 128-Bit Data Path Extension 157
Fig 58	Connector E Pinout for 192 I/O 158
Fig 59	Connector E Pinout for 128-Bit Extension and 80 I/O.....159
Fig 60	Profile A Power Connector Pin Assignment.....160
Fig 61	Profile B Power Rails.....177
Fig 62	Profile B Power Supply Sequencing.....177
Fig 63	X-Y View of Subrack and Module Relationship.....186
Fig 64	Y-Z View of Subrack with Inspection Dimension to the Bottom of the Installed Connector 187

FIGURES	PAGE
Fig 65	Y-Z View of Plug-in Unit..... 189
Fig 66	X-Y View of Plug-in Unit..... 190
Fig 67	Recommended Injector/Ejector Lever for Profile B..... 191
Fig 68	Intermodule Separation Planes, Clearances, and Maximum Component Heights, X-Z View 192
Fig 69	Profile B Front Panel Organization 194
Fig 70	Module Status State Diagram 195
Fig 71	EMI/RFI Shielding Locations and Interfacing Surfaces—Preferred 197
Fig 72	EMI/RFI Shielding Locations and Interfacing Surfaces—Alternative 198
Fig 73	Connector Naming Conventions..... 200
Fig 74	Connectors B and X Pinouts for 64/32-Bit Address and Data Paths..... 201
Fig 75	Connector E Pinout for 128-Bit Extension and 80 I/O..... 202
Fig 76	Profile B Power Connector Pin Assignment..... 203
Fig 77	Connector E Pinout for Central Arbitrator Slot..... 219
Fig 78	Backplane Geographical Address Assignment 221

TABLES	
Table 1	Hybrid Addressing.....29
Table 2	Reset (RE*) Operations.....31
Table 3	Basic Node ROM Structure.....31
Table 4	Comparison of Interrupt Registers.....39
Table 5	CSR Address Space Allocation 47
Table 6	Core CSRs and Addresses.....48
Table 7	STATE_CLEAR and STATE_SET CSR Bit Descriptions 49
Table 8	NODE_IDS CSR Bit Descriptions 50
Table 9	Internal INDIRECT_ADDRESS CSR Bit Description.....52
Table 10	Internal INDIRECT_DATA CSR Bit Description 53
Table 11	SPLIT_TIMEOUT CSR Bit Description.....53
Table 12	ARGUMENT CSRs Bit Description.....54
Table 13	TEST_START CSR Bit Descriptions.....56
Table 14	TEST_STATUS CSR Bit Descriptions 58
Table 15	UNITS_BASE CSR Bit Description 61
Table 16	UNITS_BOUND CSR Bit Description 61
Table 17	MEMORY_BASE CSR Bit Description 62
Table 18	MEMORY_BOUND CSR Bit Description 63
Table 19	INTERRUPT_TARGET CSR Bit Description 63
Table 20	INTERRUPT_MASK CSR Bit Description 64
Table 21	CLOCK_VALUE CSR Bit Description.....64
Table 22	CLOCK_TICK_PERIOD CSR Bit Description.....65
Table 23	CLOCK_STROBE_ARRIVED CSR Bit Description 65
Table 24	CLOCK_STROBE CSR Bit Description.....66
Table 25	CLOCK_REFERENCE CSR Bit Description.....67
Table 26	MESSAGE_REQUEST CSR Bit Description.....67
Table 27	MESSAGE_RESPONSE CSR Bit Description 67
Table 28	ERROR_HI CSR Bit Descriptions.....68
Table 29	ERROR_LO CSR Bit Descriptions 71
Table 30	FAILING_ADDRESS CSR.....73
Table 31	BYTE_LANE_IN_ERROR CSR.....73
Table 32	DATA_HOLD CSR 73
Table 33	TAG_HOLD CSR 73
Table 34	TIME_STAMP CSR 74
Table 35	Futurebus+ Specific CSRs 75

TABLES	PAGE
Table 36	LOGICAL_COMMON_CONTROL CSR Bit Descriptions 75
Table 37	LOGICAL_MODULE_CONTROL CSR Bit Descriptions 77
Table 38	BUS_PROPAGATION_DELAY CSR Bit Descriptions 80
Table 39	COMPETITION_SETTLING_TIME CSR Bit Descriptions 80
Table 40	TRANSACTION_TIMEOUT CSR Bit Descriptions 81
Table 41	MESSAGE_PASSING_SELECT_MASK CSR Bit Descriptions 81
Table 42	BUSY_RETRY_COUNTER CSR Bit Descriptions 82
Table 43	BUSY_RETRY_DELAY CSR Bit Descriptions 83
Table 44	ERROR_RETRY_COUNTER CSR Bit Descriptions 84
Table 45	ERROR_RETRY_DELAY CSR Bit Descriptions 84
Table 46	Node ROM Specified Locations 86
Table 47	Futurebus+ Required BUS INFORMATION BLOCK ROM Entries 87
Table 48	MODULE_LOGICAL_CAPABILITY CSR Bit Descriptions 88
Table 49	NODE_CAPABILITIES_EXT CSR Bit Descriptions 91
Table 50 (a)	ROM Directory Entry Format 95
Table 50 (b)	Directory "Key Type" Definitions 95
Table 50 (c)	Directory Entry Keys 95
Table 51	NODE_CAPABILITIES CSR Bit Descriptions 98
Table 52	MEMORY_EXTENT Bit Descriptions 100
Table 53	UNITS_EXTENT Bit Descriptions 101
Table 54	Node Indicator Relationship to State 106
Table 55	Global Timing Specifications 115
Table 56	Timing Specifications for (<i>Arbitration Type</i>) Arbitration 115
Table 57	Timing Specifications for the Arbitrated Message Bus Used for Messages 115
Table 58	Timing Specifications for the Parallel Protocol 115
Table 59	Profile A Logical Layer Referenced Specifications 121
Table 60	Profile A Physical Layer Referenced Specifications 123
Table 61	CSR Address Space 128
Table 62	Profile A Core CSRs 128
Table 63	Profile A Specific CSRs 130
Table 64	Profile A ROM Registers 131
Table 65	Calculation of Maximum Component Heights 147
Table 66	Profile B Logical Layer Referenced Specifications 162
Table 67	Profile B Physical Layer Referenced Specifications 164
Table 68	Global Timing Specifications 169
Table 69	Timing Specifications for Central Arbitration 169
Table 70	Timing Specifications for the Arbitrated Message Bus Used for Decode of the Power Fail 170
Table 71	Timing Specifications for the Parallel Protocol 170
Table 72	CSR Address Space 171
Table 73	Profile B Core CSRs 172
Table 74	Profile B Futurebus+ Specific CSRs 173
Table 75	Profile B ROM Registers 174
Table 76	Backplane Characteristics 180
Table 77	Module Signal Line Characteristics 181
Table 78	Calculation of Maximum Component Heights 188
Table 79	Profile F Logical Layer Referenced Specifications 206
Table 80	Retry Delay Times 210
Table 81	Global Timing Specifications 211
Table 82	Timing Specifications for Central Arbitration 212
Table 83	Timing Specifications for Arbitrated Messages 212
Table 84	Timing Specifications for the Parallel Protocol 212

TABLES		PAGE
Table 85	Profile F Core CSRs.....	214
Table 86	Profile F Futurebus+ Specific CSRs.....	216
Table 87	Profile F ROM CSRs.....	216
Table 88	Backplane Delay Encoding on BD[2..0]*	220
Table 90	Backplane Delay Message Fields.....	221

IEEE Standard for Futurebus+ — Physical Layer and Profile Specification

1. Introduction

1.1 Scope. The initial work in development of the Futurebus+ specification was done under the auspices of the IEEE Computer Society Microprocessor Standards Committee. In 1988 both the United States Navy's Next Generation Computer Resources (NGCR) Backplane Standards Committee and the VMEbus International Trade Association (VITA) (a trade association of both VMEbus manufacturers and users) agreed to join the IEEE in revising the initial 1987 standard (IEEE Std 896.1-1987). In early 1989 the Multibus Manufacturers Group (MMG) (a trade association of both Multibus I and Multibus II manufacturers and users) also agreed to join this effort.

The primary goal of all four groups (IEEE, U. S. Navy, VITA, and MMG) was to provide a new microprocessor bus standard that would be commercially viable and that would be acceptable to the two manufacturer groups and the three user communities.

This work resulted in the IEEE 896 family of standards. IEEE Std 896.1-1991 [2]¹ defines the logical functionality of the set of signals that make up the bus. IEEE Std 896.2-1991 (this publication) describes and specifies the physical layer (electrical characteristics, pin-outs, connector locations, module sizes, etc.) required. It also contains application environment profiles. These profiles are descriptions of functional requirements with pointers to existing standards, including portions of this one, that select and bind options within those standards. It is to these profiles, not to the component standards, that manufacturers may claim conformance. An OEM or end user who then purchases modules complying to a given profile from a range of suppliers has a higher assurance of interoperability.

Three profiles are included in this standard at the time of publication. Additional profiles, that address other aspects of the Futurebus+ compute spectrum, are being developed by the working group. These will appear in companion standards. As new physical or electrical layer requirements (e.g., a different connector type or driver technology) emerge, new profiles will be developed to address the enhanced capabilities available from newer technologies; this is part of the reason for layering the Futurebus+ standards.

The scope of this standard has been restricted to exclude some of the higher level system requirements associated with bus-based computer systems; these are addressed in companion standards P896.3 [3], P896.4 [4], P896.5 [5], P896.6 [6], P896.7 [7], and P896.8 [8].

The software interface to common node capabilities as shared by Futurebus+, Serial Bus, and SCI, is defined by IEEE Std 1212-1991 [12]. This interface provides the framework for defining processor, memory, and I/O nodes on the Futurebus+, as well as bridges to other buses (for example, P1014.1 [9]).

Where appropriate, each subject of the standard is divided into two parts: a description and a specification. The descriptions are intended only as an aid to understanding operation of the bus and are not intended to imply any requirements on equipment built to this standard. The specifications contain all the requirements with which equipment designed to this standard must comply. Should any divergence exist between the descriptions and the specifications, the specifications shall take precedence.

¹ The numbers in brackets correspond to those of the references in 1.2.1.

1.2 References. The following publications should, when applicable, be used in conjunction with this standard.

1.2.1 Protocol Stack Specifications

- [1] EIA IS-64 (1991), 2 mm Two-Part Connector for Use with Printed Boards and Backplanes.²
- [2] IEEE Std 896.1-1991, IEEE Standard for Futurebus+ — Logical Protocol Specification.³
- [3] P896.3, Futurebus+ Recommended Practices (Draft No. 4, Jan. 7, 1992).⁴
- [4] P896.4, Conformance Test Requirements for Futurebus+ (Draft No. 1.1, Jul. 26, 1991).
- [5] P896.5, Futurebus+ — Military Systems, Profile M (Draft No. 2, Jan. 1992).
- [6] P896.6, Futurebus+ — Telecommunications Systems, Profile T (Draft 1.0, Jan. 1992).
- [7] P896.7, Interconnect between Futurebus+ Systems (Draft 1.0, Jan. 1992).
- [8] P896.8, Small Computer Expandability Module for Futurebus+ Systems, Profile D (Draft 0.2, Mar. 1992).
- [9] P1014.1, VME Bus, Futurebus+ Extended Architecture (VFE) (Draft No. 8, Jan. 7, 1992).
- [10] P1156.1, Environmental Specifications for Computer Modules (Draft No. 3.2, Nov. 19, 1991).
- [11] IEEE Std 1194.1-1991, IEEE Standard for Electrical Characteristics of Backplane Transceiver Logic (BTL) Interface Circuits.
- [12] IEEE Std 1212-1991, IEEE Standard for Control and Status Register (CSR) Architecture for Microcomputer Buses.
- [13] P1212.1, Configuration Status Register for Direct Memory Access (CSR/DMA) Microcomputer Buses.
- [14] IEEE Std 1301-1991, IEEE Standard for a Metric Equipment Practice for Microcomputers—Coordination Document
- [15] IEEE Std 1301.1-1991, IEEE Standard for a Metric Equipment Practice for Microcomputers—Convection-Cooled with 2 mm Connectors.
- [16] P1394, High Performance Serial Bus, Draft No. 5.0, Mar. 8, 1992.
- [17] IEEE Std 1596-1992, IEEE Standard Scalable Coherent Interface (SCI).

1.2.2 International Government Regulations for Safety and EMI/RFI Control. This list is not intended to be all-inclusive or final. The appropriate government agencies should be contacted for the latest status of these and other regulations:

² EIA publications are available from Global Engineering, 1990 M Street NW, Suite 400, Washington, DC 20036, USA.

³ IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

⁴ Numbers with a "P" prefix are IEEE-authorized standards projects. For information about obtaining drafts, contact the IEEE Computer Society, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903.

Parameter	Agency	Regulation
Safety	CSA	CSA 22.5 #950
	IEC	IEC 950
		EN 60950
	UL	UL 1950
EMC	PTT	(license)
	FCC	Part 15
		TP-5
	ZZF	VDE 0878
	CENELEC	CISPR-22 or European Norm 55022
	VCCI	CISPR-22, with 6 dB margins
ESD	PTT	(license)
	IEC	13.6.1
Radiated Susceptibility	IEC	13.6.2

2. Definitions and Structure

2.1 Special Word Usage

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

optional, option. Refers to the implementation or use of a function that is not required. Options may be Futurebus+ specific, profile-specific, vendor-specific, or unit-specific. The latter two are not discussed or specified in profiles, and may be implemented provided they do not conflict in any way with profiles or other standards, have no effect on required levels of interoperability, and are fully documented in product specifications. Futurebus+ specific options are those that are specified in Futurebus+ documents. Reference shall be made to the appropriate standard, with wording such as “. . . if implemented, this option shall meet the requirements specified in (document and section).” An example of a Futurebus+ specific option is distributed arbitration. Similarly, a profile-specific option is one that is specified in detail in the profile. Any profile-compliant design implementing that function shall do it as specified in the profile.

Availability of an option shall not be required for the correct operation of a profile-compliant module or system; modules implementing an option should be able to default to operation with the option disabled.

Availability of an option should be specified in a documented capability ROM bit in CSR; a corresponding read/write bit should be available to turn the function on or off.

prohibited. Profile-compliant systems or modules shall not exhibit or assert the behavior or function under discussion. Modules compliant with multiple profiles may implement a function prohibited by a profile provided that function is disabled in systems compliant with that profile, or provided they do not direct prohibited transactions at modules compliant with that profile.

required or mandatory. All profile-compliant modules or systems shall implement the specified feature or function.

shall. A keyword that indicates a mandatory requirement. Designers must implement all such mandatory requirements to ensure interoperability with other compliant designs.

should. A keyword that indicates flexibility of choice with a strongly preferred implementation. The phrase *it is recommended* is used interchangeably with the keyword *should*.

supported. Profile-compliant systems or modules that respond appropriately to the existence or assertion of the specified feature or function. The profile or a referenced Futurebus+ standard shall specify the appropriate functional behavior or design.

not supported. Profile-compliant systems or modules that do not respond, or that respond inappropriately, to the existence or assertion of the feature under discussion.

2.2 Definitions

activate. The action of applying signals to a group of bus lines. Similarly, the term activated is used to describe the state of a group of bus lines when they carry signals.

application environment profile (AEP). A document that describes functional requirements and points to existing standards, selecting and binding options within those standards. An implementor who then designs a specific module and/or system should be reasonably assured that another designer's (manufacturer's or supplier's) modules will properly function within the same system. This includes all aspects of definition: mechanical, electrical, protocol, environmental, and system considerations.

assert. The action of applying a logic one signal to a bus line. Similarly, the term asserted is used to describe the state of a bus line when the signal it carries represents a logic one.

*** (asterisk).** When appended to a signal's name, the suffix "*" indicates that the logic one state of the signal is such that it will override the logic zero state applied by any other module on that line.

backplane. An electronic circuit board and connectors used to interconnect modules together electrically. The backplane connects selected pins of the connectors, thus providing the medium for the transfer of signals needed for the operation of the bus.

backplane bus. A means to connect circuit modules using common signal traces on a backplane and a standard set of rules.

backplane capacity. The minimum and maximum number of backplane slots permitted.

BIST. Built-in self-test.

BTL. Bus transceiver logic.

bus bridge. An interconnect between two or more buses, which provides signal and protocol translation from one bus to another. The buses may adhere to different bus standards for mechanical, electrical, and logical operation (such as a bus bridge from Futurebus+ to VMEbus or to Multibus II). (See also: **system bus bridge**.)

bus line. The medium for the transmission of signals. Since Futurebus+ uses open collector drivers, a bus line may be driven by several boards simultaneously. Therefore, the signal carried by the bus line is the combination of signals applied to that line from each board.

command_reset. An initialization event that is initiated by a write to the RESET_START CSR.

CSR. Control and status registers.

delta_V_{gnd}. Voltage differential of 0 V dc rail between any two worst-case points of use on the backplane, including terminating resistors and backplane capacitors.

failover. The process of reconfiguration after a fault or for planned maintenance. Failover may be manual or automatic.

geographical address. A unique identifier statically assigned to each slot by the backplane.

glyph. A picture, logo, or symbol; used instead of text.

glitch filter. As explained in IEEE Std 896.1-1991 [2], filters out a fundamental transmission line effect found in bused backplane implementations. The effect is commonly called the *wire-OR glitch*.

immediate entry. A ROM entry that provides a 24-bit immediate data value.

intermateable. Mechanical compatibility between modules with the backplane, card cage, and system into which they are inserted. Sometimes includes compatibility in the assignment of power and ground pins.

interoperate, interoperability. Compatibility of modules with each other or with the system/backplane into which they are inserted. When a pair of such units is said to *interoperate*, (a) they cannot suffer damage as a consequence of being powered and functioning in the same system; (b) the modules and the system will each be able to perform the basic function for which they were designed; (c) the modules will be able to communicate with each other using specified Futurebus+ transactions.

Some modules or backplanes will have optional features not shared by others in the same system, so they need to be able to default to the smaller common set in order to communicate with each other. A module or system working in such a reduced-capability mode may suffer performance degradation that may be severe, but will still be interoperable in that system.

key. When used in the context of a ROM entry, refers to an 8-bit field whose value identifies a ROM location as an immediate entry, offset entry, leaf entry, or subdirectory entry. This is a term used (but not defined) in IEEE Std 1212-1991 [12].

leaf entry. A ROM entry that specifies the address of a variable-length data block.

little addressian. A term used to describe the physical location of data-byte addresses on a multiplexed address/data bus. On a little addressian bus, the data byte with the smallest address is multiplexed (in time or space) with the least significant byte of the address.

live insertion. The process of inserting boards into a backplane when power is on.

live insertion state. A state in which a module may be removed from a powered, operational system without interfering with the operation of the remainder of the system or causing safety hazard to the maintenance operator.

live removal. The process of removing boards from a backplane when power is on.

LSB. Least significant bit.

L_{stub}. Stub length, measured from the connector via to the center of the transceiver pin surface-mount pad (include etch length from dispersion via, if present).

master or reference clock node. The node managing the overall synchronization process, which initiates clock synchronization cycles.

module. An electronic circuit assembly that connects to one or more slots on the backplane. It is removable from and replaceable in a backplane assembly via connectors.

monarch processor. The processor that is selected to partially initialize the local-bus resources and fetch the initial boot code.

MSB. Most significant bit.

node. A node is a set of control and status register addresses (including identification-ROM and reset command registers) that are initially defined in a 4 kbyte (minimum)

initial node address space. Each node can be reset independently (a reset of one node has no effect on other nodes).

Node ROM. A range of register addresses that are mapped to address offsets 1024 to 2047. In IEEE Std 1212-1991 [12] this is known as ROM_WINDOW.

offset entry. A ROM entry that provides a 24-bit offset value. The offset value specifies the location of a CSR that provides a 32-bit parameter.

profile. See: **application environment profile (AEP).**

Profile A module. The assembled unit, containing a Futurebus+ interface and one or more nodes complying with Profile A, that is inserted into a compatible Profile A slot. Profile A modules may operate in systems complying with other profiles if the system meets Profile A physical requirements and if modules support a compatible transaction set when sharing data.

Profile A system. The assembly of hardware elements made up of, at a minimum, the Profile A compliant backplane and subrack, power supply, fans, etc. Modules complying with other profiles may operate compatibly with Profile A systems and modules if they meet Profile A physical requirements and if their features constitute an identity or a superset of those implemented in a Profile A system as per Tables 59 and 60.

Profile B module. The assembled plug-in unit, containing a Futurebus+ interface and one or two nodes complying with Profile B, which is inserted into a compatible Futurebus+ slot. Profile B modules may operate compatibly in systems complying with other profiles if the system meets Profile B mechanical requirements, and if non-Profile B nodes properly subset their transaction set when addressing Profile B modules, as specified in this profile.

Profile B system. The assembly of hardware elements made up of, at a minimum, the Profile B compliant backplane and card cage, power supply, air mover, and a bridge to the rest of the system or to another bus. Modules complying with other profiles may operate compatibly with Profile B systems and modules if they meet Profile B mechanical requirements, and if their features constitute an identity or a superset of those mandated in this profile.

Profile F module. The assembled unit, containing a Futurebus+ interface and one or two nodes complying with Profile F, which is inserted into a compatible Futurebus+ slot. Profile F modules may operate compatibly in systems complying with other profiles if the system meets Profile F mechanical requirements, and if non-Profile F nodes properly subset their transaction set when addressing Profile F modules, as specified in this profile.

Profile F system. The assembly of hardware elements made up of, at a minimum, the Profile F compliant backplane and card cage, power supply, and an air mover. Modules complying with other profiles may operate compatibly with Profile F systems and modules if they meet Profile F mechanical requirements, and if their features constitute an identity or a superset of those mandated in this profile.

quiescent state. A state in which all modules on a Futurebus+ backplane bus have ceased all activity on all Futurebus+ defined signals and all user I/O field signals.

release. The action of applying a logic zero signal to a bus line. Similarly, the term released is used to describe the state of a bus line when the signal it carries represents a logic zero.

request message. A message that generates one or more response messages when processed.

root directory. A region in ROM specified in IEEE Std 1212-1991 [12] whose size is identified in the first location of the root directory and whose contents include ROM entries that may be identified using the ROM key.

R_{sig}. Signal etch resistance.

R_{term}. Terminator resistance (shall be placed on the backplane).

SELV. Safety extra low voltage.

serial bus. Intended as a low-cost peripheral connect or an alternate diagnostic and control path. One instantiation of a serial bus is the "Serial Bus" as specified in P1394 [16].

signal names. Where a group of bus lines are represented by the same letters, the lines within the group are numbered, e.g., **AD0***, **AD1***, **AD2***, etc. In order to represent a group of lines or signals in more convenient form, the notation **AD[31..0]*** is used. Also, the notation **AD[]*** is used to refer to all of the lines within the group.

signal skew. The time difference a data bus may have relative to its corresponding strobe signal. Signal skew is a positive quantity when the strobe signal delay exceeds the bus signal delay.

slot. A mechanical location on a backplane.

stub. The signal path on the module connecting the BTL transceiver to the Futurebus+ connector.

subdirectory entry. A ROM entry that specifies the address of another ROM subdirectory.

synchronization internal. The time period between clock synchronization cycles.

system bus bridge. The interface between a Profile B Futurebus+ and the system CPU/main memory which uses Futurebus+ to communicate to I/O subsystems. Typically, the bridge couples the Profile B I/O bus to an internal system bus, which links CPU and main memory.

A distinction should be made between the two types of bus bridges defined in this section: the *bus bridge* must comply with Profile B requirements in all respects, while the *system bus bridge* may be exempt from certain Profile requirements, such as accessibility of its CSRs from the Futurebus+, and mechanical requirements. Where different or reduced constraints apply to system bus bridges, they are called out in the appropriate section of the profile.

system configuration process. The software that initializes the system. The monarch processor executes the system configuration process.

system reset. As specified in IEEE Std 896.1-1991 [2], this is an initialization event initiated when any module asserts the **re*** signal for 100 to 200 ms. The equivalent term for this event, found in IEEE Std 1212-1991 [12], is *power_reset*.

V_{term}. Terminator source voltage to be maintained under all load conditions.

unit. A subcomponent of a node that provides a processing, memory, or I/O functionality. After the node has been initialized (typically by generic software), the unit provides the register interface that is accessed by I/O driver software. The units normally operate independently of each other, and do not affect the operation of the node upon which they reside. Note that one node could have multiple units (for example, processor, memory, and SCSI controller).

write one to clear. A method used to clear specific bits in a register. For example, if a write one to clear register contained 0xFFFFFFFF and the value 0x00800000 was written into it, the contents of the register would become 0xFF7FFFFFFF.

Z₀. Characteristic impedance of signal lines, without including connector vias, on the unpopulated backplane.

Z_0 . Characteristic impedance of signal lines on backplane populated by male connectors, without termination resistors and with no modules inserted.

Z_{stub} . Characteristic impedance of signal line stub on the module.

2.3 Signal Conventions. Figure 1 shows an example of the signal conventions used in this document. The example is shown with open collector technology (although this specification is independent of this logic family). The signal that a module applies to the input of its driver is designated by a lowercase label, e.g., **ai**. The signal that a specific module applies to a bus line is designated by a lowercase label with an asterisk, e.g., **ai***. The signal that appears on a bus line as the result of the combined signals applied to it by all modules is designated by an uppercase label with an asterisk, e.g., **AI***.

When appended to a signal name, the suffix “f” (filtered) refers to the bus line signal after it has passed through a receiver and a wired-OR glitch filter (integrator). For example, **AI_f** refers to the signal on the **AI*** line, after it has passed through an inverting receiver to become **AI** and the wired-OR glitch filter to become **AI_f**.

In a label with an appended asterisk (e.g., **WR***), a “0” would mean the signal was released. A “1” would mean an asserted signal.

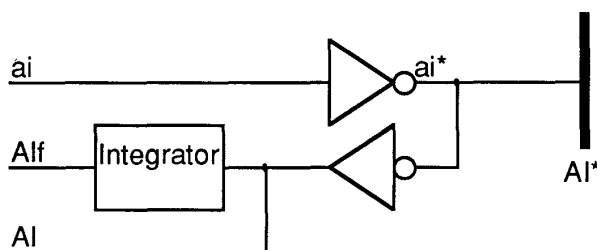


Fig 1
Signal Conventions

2.4 Numbering Conventions

- Hexadecimal numbers in this specification are represented by the notation 0x####, where each “#” is an integer from 0–9 or a letter from A–F.
- Binary numbers in this specification are represented by a sequence of 1’s and 0’s with subscript 2 (e.g., 10 = 0xA = 1010₂).
- All other numbers in the specification are decimal numbers.
- Superscript (e.g., 2³²) indicates “to the power of.”
- All register names and bit names are uppercase.
- For all CSRs in this specification the bit position of each data bit when reading or writing the CSR from the Futurebus+ may be determined by overlaying the following figure on the CSR figure.

Fig 2 Numbering Conventions

Fig 3
IEEE 896 Logo

3. Node Management

3.1 Introduction. Node management operations include address allocation, configuration control, clock synchronization, error reporting, test, communication, and generic control. Some of these operations are accomplished through standard control and status registers (CSRs). Many of the CSRs required for Futurebus+ are specified in Section 7 of IEEE Std 896.1-1991 [2], or in the IEEE Std 1212-1991 [12]. A number of fields defined as bus-dependent or optional in IEEE Std 1212-1991 are specified in this section. This section also specifies CSRs and address offsets that are not specified elsewhere. In the case of conflict in the specification of any register, IEEE Std 896.1-1991 takes precedence over this document.

CSRs are used to define module characteristics in a uniform way for buses such as Futurebus+, Serial Bus (P1394 [16]) and SCI (P1596 [17]). The CSRs appear as memory-mapped registers that are used to control node initialization (booting), identification, configuration, and diagnostics. For example, Futurebus+ modules use the CSRs to perform the personality-defining functions that VMEbus modules typically do with jumpers.

This section introduces the use of arbitrated messages for interrupts, support for the node's use of available address space, CSR addressing, error handling, clock synchronization, and node control. The intent is to define interfaces, not specify algorithms and implementation.

For the purposes of this section, a node is a set of CSRs, typically including a processor with memory, which functions independently of actions on other sets of CSRs. In Futurebus+, a module, which may occupy more than one physical board, may contain one or two nodes, for a total of 62 addressable nodes in a single Futurebus+ subrack. The module is the entity that is removed from or inserted into a Futurebus+ subrack as a unit.

Futurebus+ signaling is defined between modules; however, address and data lines (AD[63..0] and D[255..64]) actually transmit addresses and data between nodes. The relationship between boards, modules, and nodes is shown in Fig 4. The figure is not meant to imply an implementation. CSRs for different nodes on the same module may or may not be comprised of separate physical entities.

3.1.1 Address Space

3.1.1.1 Memory Addresses. Futurebus+ optionally supports 64-bit addressing and is required to support 32-bit addressing. The 32-bit address space, including the space reserved for CSRs, is illustrated in Fig 5(a). The 64-bit address space is unstructured in the sense that Futurebus+ does not reserve any areas in it. However, IEEE Std 1212-1991 [12] reserves the top 256 Mbytes of the 64-bit address space.

3.1.1.2 CSR Addresses. As shown in Fig 5(a), CSRs are addressed when the most significant four bits of the 32-bit address are asserted. The CSR architecture requires 1/16 of the available 32-bit address space.

When referencing CSRs, 32-bit addresses (A32) shall be used. Supported transactions and capabilities for CSR references are specified in 3.2. D32 transactions are always supported in the CSR area. 64-bit data transfers (D64), 128-bit data transfers (D128), or 256-bit data transfers (D256) are permitted.

Node addressing allows 1023 separate buses, each with 63 nodes, to co-exist in the same address space. This makes seamless references to nodes on various buses possible. A "bus" may be a bus other than Futurebus+, providing a bus bridge exists. One geographical address, 0x1F, is reserved, so a Futurebus+ bus is limited to 62 nodes.

The nodes on the same Futurebus+ bus may access each other's CSR space by using bus address 1023 or the actual bus number. The alias bus number, 1023, simplifies addressing on a local bus. The actual bus number, stored in the NODE_IDS CSR, is important to any nodes that may generate a split response to a requester on the far side of a bridge. The importance of the actual bus number is that the split responder will include it in the address

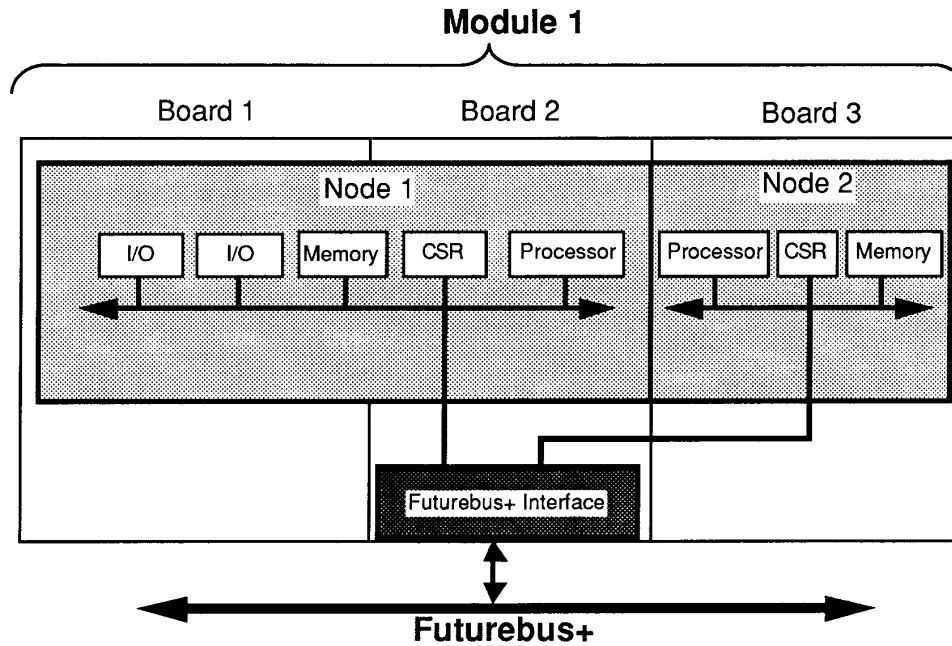


Fig 4
Futurebus+ Boards, Modules, and Nodes

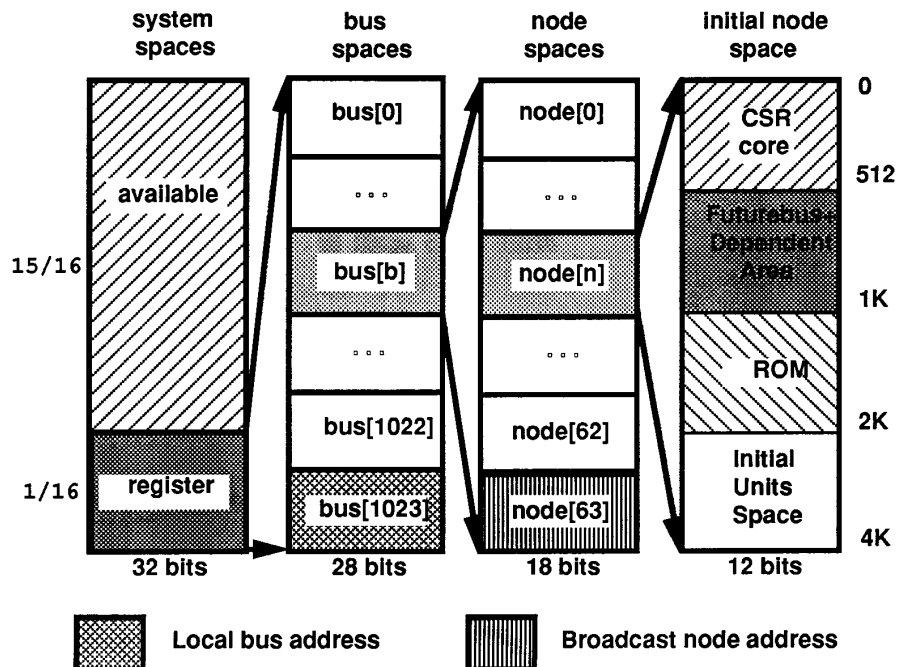


Fig 5(a)
32-bit CSR Addressing

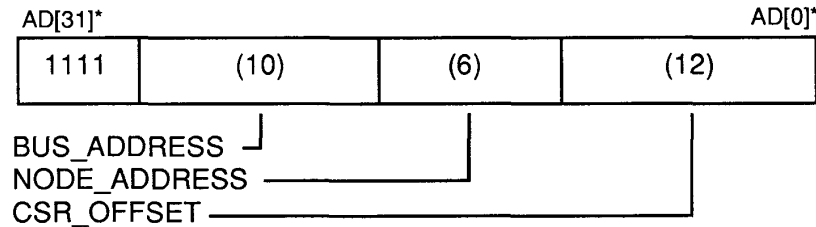


Fig 5(b)
32-bit CSR Addressing

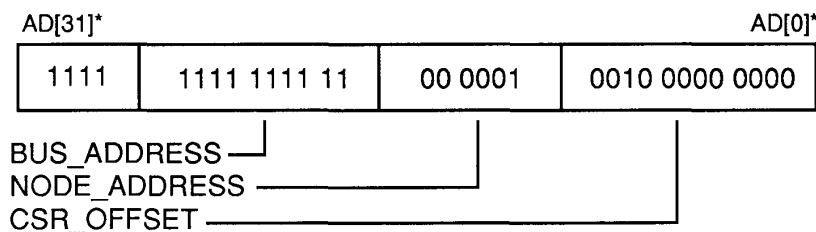


Fig 5(c)
First CSR Address of Local Bus Node One

during the connection phase of an uncached split response, and in the data during the disconnection phase of a cached split response. Masters may substitute the local **BUS_ADDRESS** alias (1023) when addressing a node on the local bus.

Since the default address of the local bus is 1023, initial access to nodes should use **BUS_ADDRESS** 1023. The most significant five bits of the node address are taken from the geographical address lines (**GA[4..0]***). Since a Futurebus+ module can never contain more than two nodes, the least significant bit of the node address is assigned by the board vendor.

A broadcast to all nodes on a Futurebus+ backplane is achieved by addressing node 63 on bus 1023. A broadcast directed to any remote bus may be accomplished by addressing node 63 and the address of the target bus. Modules participating in the broadcast transaction will set the module transaction attribute **BROADCAST**,⁵ which results in the assertion of **ST3*** (also called broadcast, **BC***).

Figure 5(b) describes CSR address space (the same information as Fig 5(a)) in an alternate format. Figure 5(c) shows the address of the first CSR in the Futurebus+ reserved area on the local bus for node 1.

3.1.1.3 Hybrid Addressing. Futurebus+ nodes must support A32 and may support A64 (64-bit addressing). Futurebus+ modules that support only A32 can coexist on the same backplane with modules that are using A64 without fear of aliasing. To support hybrid systems, since all Futurebus+ modules are required to support A32, all transactions with 32-bit equivalents shall be done with A32 transactions. A node that is A64 internally shall be designed to recognize a bus address of less than 3.75 Gbyte (the bottom 15/16 of 32-bit address space) as shown in Table 1, and convert it to A32. This slightly increases the cost of 64-bit nodes, but ensures compatibility with the elementary 32-bit-only designs.

⁵ Specified in IEEE Std 896.1-1991 [2], Section 6.

The lower 15/16 of the 32-bit address space maps directly into the 64-bit address space. The combination of address width (**CM7***) and address signals driven on the AD bus determine the target of a transaction, as shown in Table 1.

Table 1
Hybrid Addressing

CM7*	ADDRESS FIELD FROM TO		DESCRIPTION
0	0000 0000	FFFF FFFF	A32 Memory Address Space
0	F000 0000	FFFF FFFF	Node CSR Space
1	0000 0000 0000 0000	0000 0000 EFFF FFFF	PROHIBITED (Use the A32 Equivalent)
1	0000 0000 F000 0000	FFFF FFFF FFFF FFFF	A64 Memory Address Space

3.1.1.4 Byte Lane Mapping. Futurebus+ systems provide address invariance when mapping between nodes with different internal byte ordering conventions by adopting a “little addressan” byte labeling convention. No matter what the internal byte ordering convention is for a particular node or module, data byte 0 always appears on **AD[7..0]***, data byte 1 appears on **AD[15..8]***, and subsequent bytes follow the same sequence.

Individual unit architectures may specify any floating point data format, DMA control structure format, or endian ordering convention. Units that expect to share data in other than the unit's native format may support more than one format. However, the selection of which format to use is a unit architecture issue.

Units in a Futurebus+ system that have different native format conventions, and expect to share integer, floating point, or any other data type, must agree on the format of any data that is to be shared (one of the units may have to reformat data). With the exception of the CSRs specified in IEEE Std 1212-1991 [12] and in IEEE Std 896.1-1991 [2], data format is beyond the scope of this standard.

A big endian convention is used in all CSR data formats specified in this standard. The most significant byte in a big endian system is byte 0, which appears as a data byte on **AD[7..0]***. This means that in CSRs which contain addresses (integers) stored as data, the most significant byte of the stored address is byte 0 and the least significant byte of stored address is byte 3. When a little endian processor accesses a CSR specified in this standard, then byte lane swapping is always necessary.

When driving an address on the bus, the least significant bit of the address is always driven on **AD[0]*** and address bit significance increases in ascending order starting at **AD[0]***. Figures 6(a) and 6(b) show how data is mapped onto the bus.

For all CSRs in this section, the bit position of each data bit when reading or writing the CSR from the Futurebus+ may be determined by overlaying Fig 2 on the CSR figure.

P896.3 [3] contains a complete description of byte ordering.

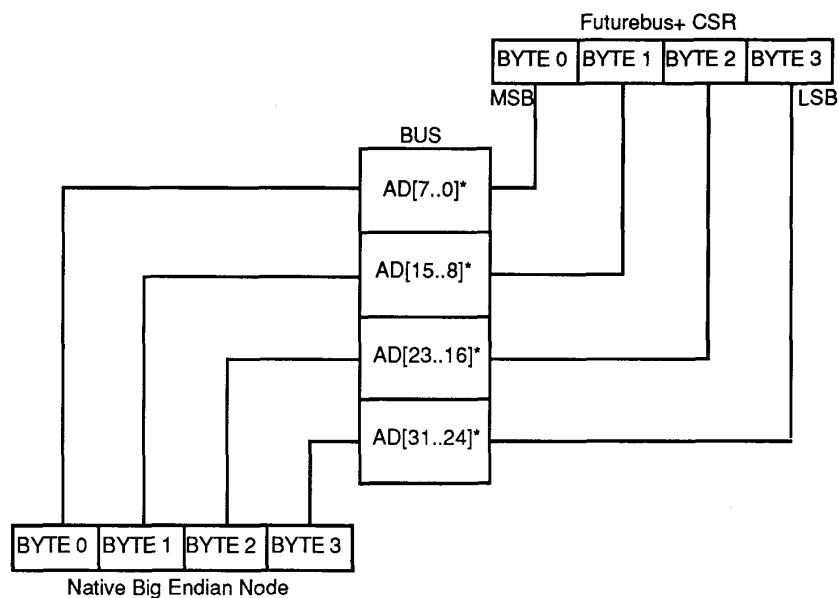


Fig 6 (a)
Native Big Endian Node to Futurebus+ CSR Byte Lane Mapping

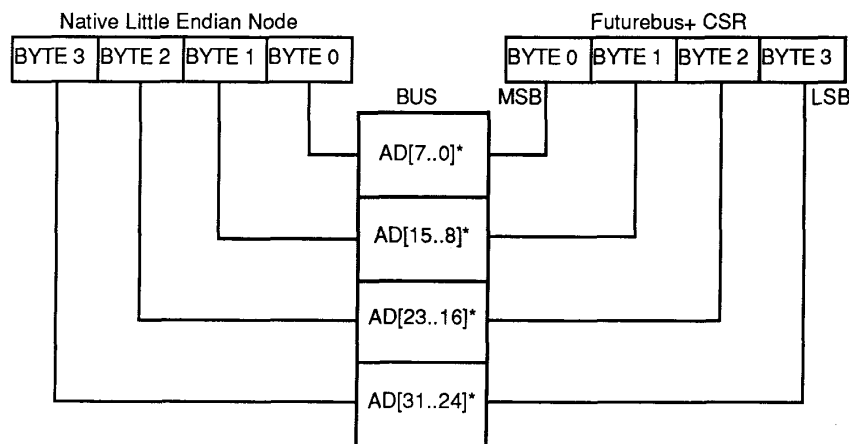


Fig 6 (b)
Native Little Endian Node to Futurebus+ CSR Byte Lane Mapping

3.1.2 Base Functional Character. The CSRs defined in this section are intended to be the minimum set necessary to “make a system work.” Operations and registers that are needed only for specific applications or only with specific system architecture paradigms are described in 3.1.3.

3.1.2.1 Resets. The three uses of the reset line specified in Section 7 of IEEE Std 896.1-1991 [2] are live insertion alignment, Bus Initialize, and System Reset. The term “command_reset” means that the identified node goes through the same reset operation that it does when it receives a system reset. Different reset events and their associated effects are summarized in Table 2.

Table 2
Reset (RE*) Operations

EVENT	EFFECT
live insertion alignment	Align only
Bus Initialize	Module bus interface only
System Reset	Entire Futurebus+ backplane
power-up	Entire Futurebus+ backplane
command_reset	Identified node

A directed or broadcast write to the RESET_START CSR initiates a command_reset. Nodes shall stay aligned to the bus throughout a command_reset and are prohibited from driving RE* as a side effect of the command_reset.

3.1.2.2 ROM Identification. The initial register space of a node includes 1 kbyte of ROM at an address offset range of 1024 to 2044. All addresses used to access the ROM should have the two least significant bits equal to zero. The ROM is used under normal operation as a storage location for read-only data or code. The information stored in the ROM is expected to be used by the system configuration process after a system reset, command_reset, or live insertion. For example, the ROM will contain data concerning the capabilities of the node, the software and hardware version of the node, and the amount of node memory that may be mapped into Futurebus+ available address space. The overall format of the ROM is specified in IEEE Std 1212-1991 [12].

The first entry in ROM is specified as required by IEEE Std 1212-1991. Futurebus+ specifies or reserves 15 locations of the ROM (each location is four bytes wide). The remaining 240 locations (960 bytes) are specified in IEEE Std 1212-1991 as available for root directory entries, unit subdirectories, root and unit leaves, and vendor-dependent information. Details of the ROM contents can be found in 3.2.4. The basic structure of the ROM is shown in Table 3.

Table 3
Basic Node ROM Structure

INFO_LENGTH	CRC_LENGTH	ROM_CRC_VALUE
Bus Information Block		
Root Directory		
Unit Subdirectories		
Root and Unit Leaves		
Vendor-Dependent Information		

The first entry in the ROM as shown in Table 3 contains three fields: The first field (INFO_LENGTH) is used to indicate the number of four-byte locations contained in the Bus Information Block. The INFO_LENGTH field will always contain a value of 15 on Futurebus+ systems. The CRC_LENGTH field indicates the number of four-byte locations protected by the value in the ROM_CRC_field. For example, if the CRC_LENGTH field contained 0xFF (255), the entire ROM (except for the first location) would be protected by the value in the ROM_CRC_VALUE field.

The next portion of the ROM is the Bus Information Block. Generally speaking, this block contains information that identifies the module as a Futurebus+ product, lists application environment profile compatibility data, and contains bus-dependent ROM entries.

The Root Directory area of the ROM contains entries that describe the module, node, and unit(s). The area also provides pointers to additional information contained in the ROM. The Root Directory might point to other subdirectories (Unit Subdirectory block) or it might point directly to leaves (Root and Unit Leaves block), which contain the actual data.

The last portion of the ROM structure is the Vendor-Dependent Information. This block contains parameters that may vary among vendors supplying the same node or unit architectures.

3.1.2.3 Capability and Setup. This section describes the capability CSRs (in ROM) and the corresponding setup (capability enable) CSRs. CSRs for bus control and arbitration are discussed in other sections.

The static characteristics and capabilities of a Futurebus+ node are stored in the ROM CSRs. This capability information is expected to be collected and used by a system configuration process. The combination of the system configuration process, software resident on the node, and system events (e.g., faults) set the dynamic characteristics of a node. The dynamic characteristics are in writable CSRs. The MODULE_LOGICAL_CAPABILITY, NODE_CAPABILITIES, and NODE_CAPABILITIES_EXT CSRs define the static capabilities of a node. Most of the remaining ROM CSRs are discussed later and relate to bus control and arbitrated messaging.

Certain node state and other status information that is referenced at system configuration time is available in the STATE_SET or STATE_CLEAR CSRs.

The LOGICAL_COMMON_CONTROL CSR may be a broadcast target and all nodes are expected to have the same capabilities enabled in this register. The LOGICAL_MODULE_CONTROL CSR is expected to be a directed write target, and it is likely that different nodes in the same system will be initialized with different capabilities enabled in this register.

The MODULE_LOGICAL_CAPABILITY ROM CSR is specified in IEEE Std 896.1-1991 [2]. Capabilities specified include support for cache coherence, split transactions, packet mode, tags, lock operations, and address and data width.

The NODE_CAPABILITIES_EXT ROM CSR is specified in 3.2.5.3. Capabilities specified in this CSR are node characteristics and indicate support for node control functions such as 64-bit address and message passing, compelled data transfers of various widths, clock synchronization, and live insertion.

The system configuration process, using the capabilities values from all nodes, will set the system characteristics. These will primarily be set in the LOGICAL_COMMON_CONTROL CSR and the LOGICAL_MODULE_CONTROL CSR. The LOGICAL_COMMON_CONTROL and LOGICAL_MODULE_CONTROL CSRs are specified in IEEE Std 896.1-1991 [2].

3.1.2.4 Bus Control. The BUS_PROPAGATION_DELAY CSR is a CSR that provides programmable control for glitch filter settling time registers. The system configuration process is expected to load this CSR with a value that is a function of the system configuration.

3.1.2.5 Distributed Arbitration. The parameter that controls distributed arbitration priority is loaded into the PRIORITY field of the NODE_IDS CSR. The COMPETITION_SETTLING_TIME CSR provides control over competition settling time. The system configuration process will load these registers with values that are a function of the system configuration. These values are typically determined at system configuration or boot time as a function of the invariant characteristics of the nodes and other characteristics of the system. When these values are changed the new value will take effect on the transition of the arbitration enable state bit (from disabled to enabled).

Control of the COMPETITION_SETTLING_TIME and PRIORITY fields may be left to the unit developer.

Modules capable of both central and distributed arbitration may be commanded to change from one to the other with a CSR write. Switching from central arbitration to distributed arbitration or distributed to central may be accomplished by clearing or setting the CENTRAL_ARBITER bit in the LOGICAL_COMMON_CONTROL CSR of all nodes on the bus. A method using arbitration messages to cause all nodes in a system to switch from one to the other is specified later in this section.

3.1.2.6 Timeout and Setup. There are several timeouts supported by the Futurebus+ standard, some have programmable values and others are specified in IEEE Std 896.1-1991 [2] as fixed.

3.1.2.6.1 Arbitration Timeout Error. ARBITRATION_TIMEOUT is specified in Section 5 of IEEE Std 896.1-1991 [2]. The timeout period is not programmable. A simplified view is that an ARBITRATION_TIMEOUT occurs only in distributed arbitration phase 2 or 4. A 1 μ s timer is started at the beginning of phase 2 and phase 4. If the timer expires before the transition to the next phase, then an ARBITRATION_TIMEOUT has occurred and the ARB_TIMEOUT_ERROR bit in the ERROR_LO CSR should be set.

3.1.2.6.2 Transaction Timeout. There is a 32-bit programmable field located in the TRANSACTION_TIMEOUT CSR. This register has a default value of 0x80000 (524288), which is approximately 122 μ s. It increments in steps of 2^{-32} (~233 ps)⁶ up to one second. The TRANSACTION_TIMEOUT CSR is used by the current master only. It guards against a parallel protocol violation causing the bus to hang when one node fails to handshake the connection, data, or disconnection phase status signals. The timeout is also used as protection against transaction times that are considerably longer than expected by the system configuration process. The value to be loaded into the TRANSACTION_TIMEOUT CSR is configuration-dependent. It may be acceptable to leave it at its default.

If the timer expires while waiting for the following phase-dependent attributes:

- Connection phase — waiting for **AK*** & **~AI***
- Data phase — waiting for **DATA_INFO_CAPTURED**
- Disconnection phase — waiting for **TRANSACTION_COMPLETE**

then an error has occurred and the TRANSACTION_TIMEOUT bit in the ERROR_HI CSR should be set.

3.1.2.6.3 Split Timeout. There is one split transaction timeout value stored across two SPLIT_TIMEOUT CSRs. The first CSR represents integer seconds and may not be required in most system configurations. The second is in fractional seconds and should be implemented on all split requestors. If this timeout counter is not implemented, then the delay waiting for a split response may be infinite. It is recommended that if there are two nodes on the same module, a single split transaction timeout value be used. A module with the capability of supporting multiple split transactions should initialize separate timers with the value in this CSR for each split transaction.

⁶ As specified in IEEE Std 896.1-1991 [2], Section 6.

If the timer expires before the response is received, the requester shall assume that the transaction is in error, and the SPLIT_TIMEOUT bit in the ERROR_HI CSR should be set.

3.1.2.7 Retry. A general retry mechanism has been provided for those systems wishing to retry a transaction that has resulted in a BUSY or an ERROR. Two sets of corresponding registers have been defined for this purpose. Each set consists of four registers as follows:

Busy Retry Registers	Error Retry Registers
BUSY_RETRY_COUNTER	ERROR_RETRY_COUNTER
BUSY_RETRY_DELAY	ERROR_RETRY_DELAY
BUSY_RETRY_COUNTER_CAPABILITY	ERROR_RETRY_COUNTER_CAPABILITY
BUSY_RETRY_DELAY_CAPABILITY	ERROR_RETRY_DELAY_CAPABILITY

A system may choose to disable all retries. This can be accomplished by setting the threshold count to zero. The details of the retry facility are left to the designer. The retry delay interval does not have to be a constant time and may be changed by the module on each retry attempt. The designer should note that certain reads may have side effects that should be considered when defining the hardware retry mechanism.

It should be noted that software retry is an alternative to implementing a hardware retry mechanism. The effectiveness of retry is completely dependent on the reason for the error or busy condition.

In general, it is not safe to automatically retry any transaction that has failed. The first attempt may have had side effects (reads of unit-dependent registers or coherent locations may have side effects). For that reason, units are only expected to automatically retry a subset of transactions that are called safe-retries. However, selecting the subset of safe-retries is unit-dependent and beyond the scope of this standard.

When an error is detected, the error symptoms are always captured in the error log registers regardless of the retry strategy. If that transaction is retried and another error is detected before the error log registers have been reset, then no new state should be set in the ERROR_HI register. If a retry succeeds at any time, the RECOVERED bit should then be set in the ERROR_LO register. If the retry count is exceeded, then the ERROR_RETRY_THRESHOLD_EXCEEDED bit is set in the ERROR_LO CSR.

P896.3 [3] discusses retry in depth in the fault tolerance section.

3.1.2.8 Node State Control. Four node states are defined: initializing, running, testing, and dead. Two additional states are defined for nodes but are not meaningful to normally operating nodes: not powered and live insertion. Transitions between the four node states can be initiated by use of operator facilities on the module front panel or by bus transactions that write to the STATE_SET, RESET_START, and TEST_START CSRs.

The node state is represented in the STATE_CLEAR CSR by the 2-bit STATE field. The STATE field can be changed by writing to the RESET_START CSR, which will cause the node to undergo a command_reset. This will cause the node to enter the *initializing* state. If no errors are detected, the node will transition to the *running* state. If a fatal fault is detected, the node will enter the *dead* state. If a permanent fault is detected but the node is able to perform its assigned function, the LIMP bit in the ERROR_LO CSR will be set and the node will enter the *running* state. If a transient fault is detected but a retry of the failing transaction succeeds, the RECOVERED bit in the ERROR_LO CSR is set.

Tests can be defined such that *running* nodes can continue to run while performing the test. The *testing* state is entered when a test is started (by writing to the TEST_START CSR) that causes the node to be unable to perform its assigned function.

Live insertion is controlled by one bit in the LOGICAL_COMMON_CONTROL CSR, two bits in the STATE_CLEAR and STATE_SET CSRs, two capability bits in the NODE_CAPABILITIES_CSR, and two capability bits in the NODE_CAPABILITIES_EXT CSR. The electrical and other non-CSR related live insertion requirements are presented and specified in Section 4.

A Futurebus+ module implements one of four live insertion levels. The module's level is specified by the LIVE_INSERTION field in the NODE_CAPABILITIES_EXT CSR.

- (1) **Level 0** — live insertion is not supported.
- (2) **Level 1** — during the withdrawal or insertion operation, no parallel or arbitration bus transactions are permitted.
- (3) **Level 2** — permits a limited number of transaction types during the withdrawal and insertion operation.
- (4) **Level 3** — permits any transaction type defined for the profile to be implemented during live withdrawal and insertion operation.

For level 2, the transaction type permitted during live insertion is defined by the application profile. If a module contains two nodes, both nodes implement the same level.

Systems that do not support live insertion (e.g., for safety reasons) may disable a live insertion capable module from entering the live insertion state (indicator lights never indicate live insertion). Disabling this function is done by setting a control bit in the LOGICAL_COMMON_CONTROL CSR, which disables the front panel live insertion switch, and the ATN and OFF state bits in the STATE_CLEAR and STATE_SET CSRs.

Setting the ATN bit in the STATE_SET CSR indicates to the module that module shutdown, in preparation for a live withdrawal operation, is about to begin. The node's functional logic (one or two nodes per module) initiates the implementation-defined housekeeping operations necessary to save the node's state prior to removing power. Examples of actions that set the ATN bit are given in Section 4. Futurebus+ modules are designed so that when housekeeping is complete, power is removed from the module without requiring additional signals. The node is permitted to initiate bus transactions as a part of housekeeping.

A module can be powered down by setting the OFF bit in the STATE_SET CSR to one. Once OFF has been set the module will no longer respond to or generate Futurebus+ transactions. How power is restored to the module is beyond the scope of this specification.

If ATN has been set, but a power_reset or command_reset is received before the housekeeping operations are complete, the housekeeping operations are terminated and the node is reset. If a Bus Initialize is received after the ATN bit has been set but prior to the completion of housekeeping, the bus interface is initialized and housekeeping continues followed by power removal.

If a module contains two nodes, receiving ATN or OFF at either node causes the action at both nodes. For ATN, power is removed when the slower node completes housekeeping.

3.1.2.9 Standard Diagnostic Interface. This section describes how to make a node resident (built in) test accessible for use by generic test control software.

3.1.2.9.1 Diagnostic Facility. Diagnosing an error is often facilitated by the availability of state information captured at the time the error was detected. The region in the CSR core address space between 384 and 508 is reserved for error symptom logging. The information to be logged in the CSR core error log buffer area on Futurebus+ systems should include (but is not limited to) the address referenced in the failing transaction, the byte lane on which a parity error was detected, and state of the AD[]* bus when a connection, data, or disconnection phase error is detected.

The Futurebus+ diagnostic facility specifies 16 Core CSRs for error symptom capturing, as defined in 3.2.2.14. In addition to error symptom capturing, the standard provides a CSR interface to initiate and monitor the progress of node and bus built-in self-test (BIST) and resets. Additional information on how the Futurebus+ diagnostic facility is intended to be used will be included in the Systems Test section of P896.3 [3].

A default self-test is assumed to run when System Reset or command_reset is signaled or when one or more tests are explicitly requested. At the end of the reset period, the module must be ready to respond if addressed. If the module is addressed and self-test is not complete, the module may respond with busy status **BS***. This standard specifies a maximum time for a default built-in self-test to complete but does not specify a maximum time limit on all tests (including those not run by default).

Diagnostic tests are divided into four categories:

- (1) **Initialization** diagnostics are autonomous in that they should not depend on the correct operation of other nodes in the system or the contents of the ARGUMENT CSRs. The maximum time limit for the default set of initialization tests to execute on a defect free node is 10 seconds. By setting the maximum value, software has a reliable mechanism for detecting failures that prevent test completion. The default initialization tests may be a subset of all tests in this category.
- (2) **Extended** diagnostics depend on the availability of a memory buffer. These tests will generate bus traffic using a memory buffer that is identified in the ARGUMENT CSRs. They should detect all permanent faults on a node that remain following execution of initialization diagnostics, with the exception of logic that is untestable without operator intervention. These tests may verify attached options such as mass storage devices and controllers. The maximum time for the default set of extended tests to execute on a defect-free node is 10 seconds. The default extended tests may be a subset of all tests in this category.
- (3) **System** diagnostics are an option of the system designer. When this test category is selected, the interpretation of data passed in the TEST_STEP field or in the ARGUMENT CSRs is not specified in this standard. For example, a vendor may define a protocol for passing ASCII characters to a module resident ROM based diagnostic monitor as a mechanism for invoking unit-specific diagnostic tests. Examples of functions that system category tests may verify include timer functions, message passing, or cache coherence in shared memory systems. These may also be used to stress the bus by generating a dense sequence of transactions that consume as much of the available bus bandwidth as possible.
- (4) **Manual intervention** diagnostics are tests that require some action on the part of the operator, such as mounting scratch media, the connection of an I/O loopback connector, or manual setting of loopback on an external modem. When this test category is selected, the interpretation of data passed in the TEST_STEP field or in the ARGUMENT CSRs is not specified in this standard. Destructive testing is only permitted in the manual category (e.g., formatting a disk, or running a read/write exerciser).

3.1.2.9.2 Diagnostic Test Initialization. Diagnostic tests are initiated by power-on, System Reset, or by writing the TEST_START CSR, a member of the core set of CSRs. The TEST_START CSR contains three fields:

- (1) **TEST_STEP** — a 16-bit field that typically identifies which test sequence to run. The interpretation of the field is implementation-specific. TEST_STEP equal to 0x0000 is reserved for the default set of tests; TEST_STEP equal to 0xFFFF indicates that all tests in the category are to be run. In the system test category, a TEST_STEP may represent a message in a vendor-defined communication protocol with a unit resident diagnostic monitor program.
- (2) **TOPS** — an optional 3-bit field indicating the test should be continually looped, should loop on error, and/or should continue or halt if an error is detected. The options are not mutually exclusive (multiple bits may be set).
- (3) **CAT (category)** — a 4-bit field indicating the category of test to be run. Tests may be initialization diagnostics, extended diagnostics, system diagnostics, or manual intervention tests. A write of 0x0 to the category field causes the current test to stop. A test may be part of one or more categories. The categories are not mutually exclusive (multiple bits may be set).

NOTE: When invoking a potentially destructive manual intervention category test, the operator is responsible for taking appropriate precautions.

The ARGUMENT CSRs are available for processes outside the node to provide parameters to guide the test, or to receive parameters from the test. The parameter will be the address of a buffer in memory that an extended test may use. The ARGUMENT CSRs can also be used in system tests to pass commands to the test, present further symptom information from the test, or to indicate the memory address where further information can be found. By convention, these CSRs will be used by the extended test category to indicate the location of a 4 kbyte memory buffer. Except for extended tests, the interpretation of the contents of the ARGUMENT CSRs are implementation-dependent and may vary between diagnostics.

A typical extended test would read the ARGUMENT CSRs to obtain the test area address, read the LOGICAL_MODULE_CONTROL CSR to determine the length and width transaction to use, then write and read data patterns to the test area. This test will exercise the transceivers in both the sending and receiving module.

3.1.2.9.3 Diagnostic Test Status. A TEST_STATUS CSR in the Core CSR space is provided both to observe the progress of the test and to determine the results of the test. This CSR is set by node logic; a write to this CSR from other Futurebus+ nodes is ignored. The CSR contains four fields:

- (1) **CAT** (category) — a 4-bit field indicating whether an initialization diagnostic, extended diagnostic, system diagnostic, or manual intervention test is running. A test may be part of one or more categories.
- (2) **TEST_STEP** — a 16-bit field that identifies the test currently running or the last test run if the tests are complete. It is assumed that tests run in sequence. Thus, the TEST_STEP field value used to start the test may be automatically succeeded by other test identifiers. The interpretation of the field, i.e., the identification of the test, is implementation-specific. In the system test category, the contents of this field may be used to pass vendor-defined information.
- (3) **FRU** (field replaceable unit) — the implementation-specific identification of the failed component, if one was identified.
- (4) **TEST_STATE** — a 6-bit field that indicates whether a test passed, failed, is implemented, or is active. When running a series of tests, it is possible for multiple bits to be set. A TEST_STATE of 000100₂ (test successful) or 000000₂ (test not implemented) indicates that the node has entered the running state. This will also be indicated in the STATE_CLEAR and STATE_SET CSRs. The TEST_STATUS field is immediately updated when a test starts, detects an error, or completes. The immediate update is permitted to clear all bits and set the running state while checking to determine if a specified test is implemented by the node.

3.1.3 Optional Functional Character. This section describes functional character needed for specific unit architecture and system paradigms. For example, only real-time systems are likely to use clock synchronization. These are all areas that are less likely to be of general utility. Standard interfaces have been defined for those functions that require that the interface address and character be common for all implementations that require the specific feature.

3.1.3.1 Messages. The optional message mechanism defines two message receive mailboxes, a selective message broadcast facility (sometimes referred to as multi-cast), and variable length messages. The presence of the message facility is indicated by the MESSAGE_CAPABLE bit being set in the MODULE_LOGICAL_CAPABILITY CSR⁷ and

⁷ The MESSAGE_CAPABLE bit in the MODULE_LOGICAL_CAPABILITIES CSR is specified in IEEE Std 896.1-1991 [2]. The NODE_CAPABILITIES CSR is specified in IEEE Std 1212-1991 [12]. Both shall be set on message capable nodes.

the MS bit being set in the `NODE_CAPABILITIES` CSR. If the `MESSAGE_CAPABLE` bit indicates that the message facility exists then the implementation shall support the entire message facility. Section 9 of IEEE Std 896.1 [2] contains the general description and specification of the message facility.

The Futurebus+ message facility defines separate target addresses for request and response messages. A request message is defined as a message that generates one or more response messages when processed. A response message generates no additional messages when being processed. Two message mailboxes defined in this manner can reduce the possibility of deadlocks. An implementation is allowed to merge `MESSAGE_REQUEST` and `MESSAGE_RESPONSE` if deadlock-free operation can be maintained. Futurebus+ nodes that merge the message targets must recognize both addresses. This will retain compatibility with other buses that support message passing as defined in IEEE Std 1212-1991 [12]. Messages directed to a specific node, i.e., not node 63, must have the least significant six bits of the address equal to zero.

The message CSRs can be used as target addresses for broadcast messages. Broadcast messages are indicated by a CSR address in the range 128 to 252 directed to node 63. When a node that supports message passing detects a broadcast message, if the least significant six bits of the address are non-zero, they are used to select a bit in the node's `MESSAGE_PASSING_SELECT_MASK` CSR. If the bit is one, the message is accepted (assuming no other conditions cause it to fail). If the bit is a zero, the message is not accepted. If the least significant six bits of the address are zero, then the message is unconditionally accepted.

CSR addresses 128 to 252 are reserved for messages. The intent is to reserve 64 bytes, for each of the message targets, to aid minimum implementations in supporting the default message size (64 bytes). A performance-oriented implementation may provide sufficient buffering to support queuing multiple messages and receive messages longer than 64-bytes. The maximum frame size a node can accept is specified by the `MESSAGE_FRAME_SIZE` CSR. Messages may be transmitted 32, 64, 128, or 256 bits in parallel.

The message mechanism can be used to transmit large amounts of data but requires access to the data transfer bus. A complementary event mechanism based on arbitrated messages (which does not require the data transfer bus) is presented in 3.1.3.2.

3.1.3.2 Arbitrated Messages. Arbitrated messages and distributed arbitration messages are defined in Section 5 of IEEE Std 896.1-1991 [2] to allow nodes to signal global events across the bus without becoming master or performing data transfers. Systems with distributed arbitration may support distributed arbitration messages. Systems with central arbitration may support general arbitrated messages and central arbitrated messages.

Central arbitrated messages are messages to the central arbiter. They include a new arbitration priority and the geographical address of the node sending the message, as specified in Section 5 of IEEE Std 896.1-1991.

General arbitrated messages and distributed arbitration messages allow nodes to broadcast a message to other nodes on the same Futurebus+ segment. Some encodings of general arbitrated and distributed arbitration message fields are reserved in this standard.

A bridge to a non-Futurebus+ backplane may convert certain arbitrated messages into interrupts, particularly if the bus "on the other side" supports the `INTERRUPT_TARGET` CSR. But in general, designers need to carefully consider how arbitrated messages will be interpreted by bus bridges.

This standard reserves general arbitrated messages and distributed arbitration messages for the following purposes:

- For the event mechanism, described in the next section
- For use by vendors
- For specification by profiles
- Unmaskable Interrupt
- Power fail

3.1.3.2.1 Event Mechanism. A write to the node's INTERRUPT_TARGET CSR can be used to cause an implementation-defined interrupt in the node. The data that is written to the INTERRUPT_TARGET register is ANDed with the data in the INTERRUPT_MASK register and the result is sent to all processors on the node. A non-zero bit in the result corresponds to one of 32 interrupt priorities. This mechanism is described in 3.1.3.3.2.

In situations where the data transfer lines are unavailable, either because a long transaction is underway or because of a fault such as a "babbling transmitter," general arbitrated messages or distributed arbitration messages can be used to signal an event to a node with minimum delay. Certain of these messages are defined to set a specified bit in the INTERRUPT_TARGET CSR. The result will be ANDed with the INTERRUPT_MASK register as indicated above.

General arbitrated messages and distributed arbitration messages 0x00 to 0x1F are defined to have a one-to-one correspondence with bits in the INTERRUPT_TARGET CSR. The least significant bit is set by 0x00, the most significant bit by 0x1F, etc. These messages are local to the particular Futurebus+ segment where they are generated, i.e., they are not necessarily passed across bridges.

The use of arbitrated messages and distributed arbitration messages for event signaling follows the same rules as the generation of all arbitrated messages.

3.1.3.3 Interrupts. Interrupt requests may originate from sources within the node that services the interrupt, from external sources, other modules, DMA engines, or sources outside the local system. Either hardware or software can potentially initiate interrupts. Futurebus+ supports two interrupt mechanisms: unit-specific interrupts and unit-broadcast interrupts. These interrupt mechanisms provide the basic facilities upon which a system designer can build an application-specific interrupt-processing mechanism.

3.1.3.3.1 Unit-Specific Interrupts. The unit-specific interrupt mechanism uses a write unlocked transaction to write a 32-bit datum to a unit space register whose location is determined by mutual agreement between the interrupting and the interrupted processes.

These interrupt registers are not to be confused with the INTERRUPT_TARGET CSR specified in 3.2.2.11. Table 4 contrasts some of the differences between the two types of interrupt registers.

Table 4
Comparison of Interrupt Registers

INTERRUPT_TARGET CSR	Interrupt Register
Target for broadcast or directed writes	Target for directed writes
One per node, located at a specific CSR offset	As many as needed, located at any address in unit space
Writes shall never be busied	Writes may be busied

Although not defined by this standard, a few examples of possible interrupt register implementations are:

- A node has one or more interrupt-registers where each bit signifies an interrupt, i.e., there are 32 possible discrete interrupts per register. When a write occurs to this register a side effect is to post an interrupt corresponding to each bit set. For example, a write of the value 0x00000001 to an interrupt register will initiate the interrupt request corresponding to the least significant bit.

- The hardware behind this register should be designed to translate interrupt bits to prioritized interrupt vectors for the appropriate processor. When a bit is set the hardware will schedule the corresponding interrupt. If multiple interrupts are posted to the same bit before it is serviced, it should be counted as one interrupt. The bit should be cleared when the interrupt is serviced.
- A node may have a simple interrupt-register that can only signal one interrupt. A write of any value to this register would signal an interrupt to the node; the actual value loaded could be an encoded representation of the interrupt level vector location.
- A node may have separate interrupt registers, one for each priority level. A write of any value to one of these registers would signal an interrupt at the priority level implicit in that register.

3.1.3.3.2 Unit Broadcast Interrupts. The INTERRUPT_TARGET and INTERRUPT_MASK CSRs provide a mechanism for sending 32 prioritized interrupts to a node. Bits in the INTERRUPT_TARGET correspond to 32 priority levels with the most significant bit corresponding to the highest priority level and the least significant bit to the lowest priority level. When a write to the INTERRUPT_TARGET CSR occurs, the INTERRUPT_TARGET is ANDed with the contents of the INTERRUPT_MASK CSR. If any bit of the result is a one, an interrupt on that level is sent to the node's processors. Masked interrupts do not remain pending; writing to the INTERRUPT_MASK CSR does not cause the AND to take place. The relationship between bits in the INTERRUPT_TARGET CSR and vectoring of external interrupts by the node's processors, other than relative priority, is defined by the implementation.

When multiple bits are set as a result of the AND of INTERRUPT_MASK and INTERRUPT_TARGET, the node's logic is expected to provide the mechanism to queue the interrupts.

As an example, a process can be assigned a given priority level when executing. If an interrupt at a lower level is received, the interrupt is logged but left pending. If an interrupt at a higher level is received, the currently executing process is interrupted and a process associated with the new priority level is dispatched.

Processors that only recognize, for example, eight priority levels would never be enabled for priority levels 8 through 31. Nodes that do not use priority-based process dispatching would always have the INTERRUPT_MASK CSR set to contain a single one bit; all other bits would be zeros.

Interrupts can be initiated using arbitrated messages as presented in 3.1.3.2.

3.1.3.4 Packet Management. Packet management is discussed in detail in IEEE Std 896.1-1991 [2]. Fields in the MODULE_LOGICAL_CAPABILITY register indicate what packet-mode capabilities a node supports. Fields in the LOGICAL_MODULE_CONTROL CSR enable those functional capabilities.

3.1.3.5 Clock Synchronization. For systems that must maintain global time among multiple nodes with real-time clocks, Futurebus+ provides timing facilities for use in conjunction with a set of real-time clock CSRs. The intent is not to fully specify a particular system clock implementation, but rather to define primitive operations that support the clock synchronization techniques described by P896.3 [3].

Futurebus+ references real-time clock CSRs defined by IEEE Std 1212-1991 [12]. These include the 64-bit CLOCK_VALUE, CLOCK_TICK_PERIOD, and CLOCK_STROBE_ARRIVED CSRs. The format of these registers is defined in IEEE Std 1212-1991. Futurebus+ defines other real-time clock registers whose addresses and names are reserved by IEEE Std 1212-1991 but whose format and function are bus-specific. These include the 32-bit CLOCK_INFO0, CLOCK_INFO1, CLOCK_INFO2, and CLOCK_INFO3 CSRs. Futurebus+ defines CLOCK_STROBE as an equivalent name with CLOCK_INFO0, reserves the use of

CLOCK_INFO1, and defines CLOCK_REFERENCE_HI and CLOCK_REFERENCE_LO as equivalent names with CLOCK_INFO2 and CLOCK_INFO3, respectively. A clock strobe signal associated with the CLOCK_STROBE_ARRIVED CSR is also defined by Futurebus+.

The minimum real-time clock CSRs required to implement a distributed synchronized clock node on Futurebus+ are: CLOCK_STROBE and CLOCK_STROBE_ARRIVED. Two bits in the NODE_CAPABILITIES_EXT ROM CSR and two bits in the LOGICAL_MODULE_CONTROL CSR are also required. In addition, it is recommended that CLOCK_REFERENCE, CLOCK_VALUE and CLOCK_TICK_PERIOD CSRs be implemented to allow the following Futurebus+ operations: broadcast writes of a reference clock value to all subordinate clock nodes; Futurebus+ reads of a node's local clock; Futurebus+ reads to inform other nodes of the local clock update period.

The time facilities in each node are under control of a software process known as the clock manager. The clock manager process uses the timing facilities to (1) establish a distributed system clock configuration, (2) initialize system time, and (3) maintain system time via periodic synchronizations.

To support the range of synchronization techniques described in P896.3 [3], all clock nodes are assumed to be intelligent entities, capable of adjusting their own local clock based upon a reference time distributed periodically by a reference clock. To support clock synchronization across a bus bridge, it is further assumed that the bus bridge has its own local clock, its own clock CSRs, and a mechanism to synchronize its local clock to a reference clock on at least one side of the bridge.

3.1.3.5.1 Clock Synchronization Summary. The time period between clock synchronization cycles is the synchronization interval. The synchronization interval should be short enough to ensure that the synchronicity of clocks is adequate for the intended application but not so short as to cause synchronization to consume an excessive amount of bus bandwidth. Minimum requirements on the oscillator accuracy and update frequency of clocks are specified in 3.2.2.12.

The node managing the overall synchronization process, called the master or reference clock node, initiates clock synchronization cycles. During each synchronization cycle, the reference clock node sends a sample_clock_strobe signal to all participating clock nodes using standard Futurebus+ signals. The sample_clock_strobe signal causes all nodes to store the current value of their local clock in a local CSR. A reference clock value, called reference_clock, is then transmitted by the reference clock node to all subordinate clock nodes. Alternatively, subordinate clock nodes may perform CSR read(s) to obtain a reference clock value from a particular node or an average value from many nodes. Each node computes the time error or offset between the local and reference clock value. When all clock nodes have adjusted the time or frequency of their local clock based upon the time error, then the clock synchronization cycle is complete.

3.1.3.5.2 Local Clock Model. Each node participating in clock synchronization maintains a free-running counter called time_running. Time_running, which does not reside in CSR space, is 64-bits long and represents time as an unsigned integer such that the least significant bit represents 2^{-32} seconds (~233 picoseconds). Thus, the most significant quadlet of time_running represents seconds and the least significant quadlet represents fractions of a second. Node-local processor reads of time_running return the current time.

NOTE: All 64 bits of time_running may not be visible to application software since, in certain synchronization methods, the m least significant clock bits are manipulated by hardware to accomplish time synchronization. In this case, the clock is divided into $64 - m$ application software usable bits and m synchronization bits. Conversely, time_running may be extended beyond 64 bits to 96 or more bits in order to implement other synchronization methods. (Refer to P896.3 [3] for a description of these clock correction methods.)

To support reads of a node's free running clock over Futurebus+ a node may, as an option, implement a read-only 64-bit CLOCK_VALUE_CSR. To accurately read a 64-bit clock value, software is expected to read the CLOCK_VALUE_HI register, to read the CLOCK_VALUE_LO register, and to reread the CLOCK_VALUE_HI register. If the CLOCK_VALUE_HI register has changed, an overflow has occurred, and the CLOCK_VALUE_LO register can be assumed to be zero (a legal value within the reading interval).

The optional 64-bit CLOCK_TICK_PERIOD_CSR is used to inform other nodes of the local clock update period. The least significant bit represents 2^{-64} seconds.

3.1.3.5.3 Clock Offset Determination. Each node participating in clock synchronization contains a 32-bit CSR called CLOCK_STROBE. A Futurebus+ broadcast write to CLOCK_STROBE is interpreted by all clock nodes as the sample_clock_strobe signal. A detailed description of the timing of the sample_clock_strobe is in 3.1.3.5.5.

Along with providing the sample_clock_strobe signal, the broadcast write to CLOCK_STROBE_CSR provides a label that uniquely identifies this strobe from previous or subsequent strobes. The least significant byte contains a COUNT field that is incremented each time the node generates a strobe. This is intended to distinguish between different strobes generated by the same node. The label contains the NODE_ADDRESS. This is intended to distinguish between strobes generated by different nodes. The remaining bits of CLOCK_STROBE are reserved by Futurebus+.

On the arrival of the sample_clock_strobe, clock nodes latch their local 64-bit clock into the local 64-bit CLOCK_STROBE_ARRIVED_CSR. Sampling occurs on all enabled clock-capable nodes including the reference node which generated the strobe and any clock capable bridge which has enabled clock nodes on the remote bus. When read over Futurebus+ the CLOCK_STROBE_ARRIVED register simply returns its current value (i.e., the most recently latched value of time_running due to a sample_clock_strobe).

After sending the sample_clock_strobe, the reference node computes a new reference time. How this is done is not defined by this standard. In the simplest case, the reference node's most recently sampled clock value (the contents of CLOCK_STROBE_ARRIVED) may be used as the reference time. Alternatively, the reference node may read each participating clock's CLOCK_STROBE_ARRIVED_CSR in order to compute an averaged or weighted-average reference clock value.

Once a reference time is determined, the reference node distributes the reference time to a target 64-bit CSR on each clock node called CLOCK_REFERENCE. The broadcast is performed as follows. The most significant quadlet of reference_clock is written to CLOCK_REFERENCE_HI. Then, the least significant quadlet of reference_clock is written to CLOCK_REFERENCE_LO. The write of CLOCK_REFERENCE_LO activates the local clock manager process (e.g., by generating an interrupt).

If the CLOCK_REFERENCE_CSR is not implemented, subordinate nodes may determine the reference time by reading a designated node's CLOCK_STROBE_ARRIVED_CSR. This method requires N bus transactions to distribute the reference clock, where N is the number of clock-capable nodes on the local bus segment. Alternatively, all nodes may read each other's CLOCK_STROBE_ARRIVED_CSR in order to compile an averaged reference clock value. This method requires N^2 bus transactions to determine the clock reference value. While requiring more overhead, these alternative methods facilitate implementation of a fault-tolerant global clock. Futurebus+ profiles are responsible for specifying the method of computing and distributing the reference value.

After the reference clock value is available in the local node, the local clock manager process computes the offset value by subtracting the stored 64-bit value of CLOCK_STROBE_ARRIVED from the 64-bit reference clock value:

$$\text{Offset} = \text{CLOCK_REFERENCE} - \text{CLOCK_STROBE_ARRIVED}$$

From this definition, a positive offset indicates the subordinate clock is running slow relative to the reference clock, and a negative offset indicates the subordinate clock is running fast.

Based upon the sign and magnitude of the computed offset, a node performs the appropriate adjustment to the local time_running. Two basic clock adjustment/correction methods are possible: adjustment of the local clock phase (value) or adjustment of the local clock frequency. The method of clock adjustment is outside the scope of this standard. For a description of these methods and their relative merits consult P896.3 [3]. The synchronization interval is complete when all clock nodes have adjusted their local clocks.

3.1.3.5.4 Bridge Considerations. The Futurebus+ clock synchronization algorithm described in 3.1.3.5.3 is extensible for use in multiple bus architectures that are interconnected by bridges. To achieve synchronization of clocks across bridges, the bridge is required to be capable of acting both as a reference and subordinate clock node as previously described. In addition, the bridge must be able to forward the sample_clock_strobe to the remote bus and be able to compute and broadcast an adjusted reference time to clock nodes on this bus. If clocks on remote bus nodes are to be included in the averaging of the reference time, then the bridge must either (1) contain CSRs to allow the reference node to normalize these CLOCK_STROBE_ARRIVED CSR values, or (2) perform the normalization itself during a read of this register. At a minimum, delay of the sample_clock_strobe signal through the bridge and across any significant amount of cabling should be accounted for. The bridge standard is ultimately responsible for unambiguously specifying this functionality.

3.1.3.5.5 Detailed Clock Strobe Timing. Upon initiation of the clock synchronization procedure, the reference clock node performs a Futurebus+ broadcast write transaction to the CLOCK_STROBE CSR. All clock nodes, including the reference clock node, are to recognize that when they decode this address that the sample_clock_strobe signal is imminent. The timing diagram of this signal is shown in Fig 7. Each time-synchronized bus interface unit should not acknowledge the address, i.e., release *ai*, until it is ready to accept the sample_clock_strobe. All nodes must release *ai* in a timely manner, however, so as not to cause a bus transaction timeout.

When the reference node observes *AI** released, it asserts *ds*, causing *DS** to be asserted. This odd beat signal transition of *DS** is interpreted as sample_clock_strobe by all clock nodes. The *DS** odd-beat transition causes all participating nodes, including the reference clock node, to latch the value of the local time_running counter into the local CLOCK_STROBE_ARRIVED CSR. All nodes should be able to sample their time_running CSR within 80 ns of the *DS** odd beat transition. It is expected that well-designed modules will be able to achieve a much tighter tolerance.

3.1.3.5.6 System Clock Configuration. Nodes that participate in the clock synchronization procedure indicate whether they have reference clock capability or subordinate clock capability by having the REFERENCE_CLOCK_CAP or LOCAL_CLOCK_CAP bits set in the NODE_CAPABILITIES_EXT register. For fault-tolerant applications requiring spare reference clock nodes and/or real-time applications requiring the use of the "best available" clock as the reference clock in a given configuration, clock nodes should be implemented with both reference and subordinate clock capability.

The Futurebus+ node which is to be responsible for coordinating the overall clock synchronization process will have the REFERENCE_CLOCK bit set in the LOGICAL_MODULE_CONTROL CSR. The reference node is responsible for determining and distributing the initial reference clock value to all participating clock nodes on the local Futurebus+. The reference node is further responsible for initiating clock synchronizations at the required (profile-dependent) update rate. A node which is to participate as a subordinate clock will have the LOCAL_CLOCK bit set in its LOGICAL_MODULE_CONTROL CSR.

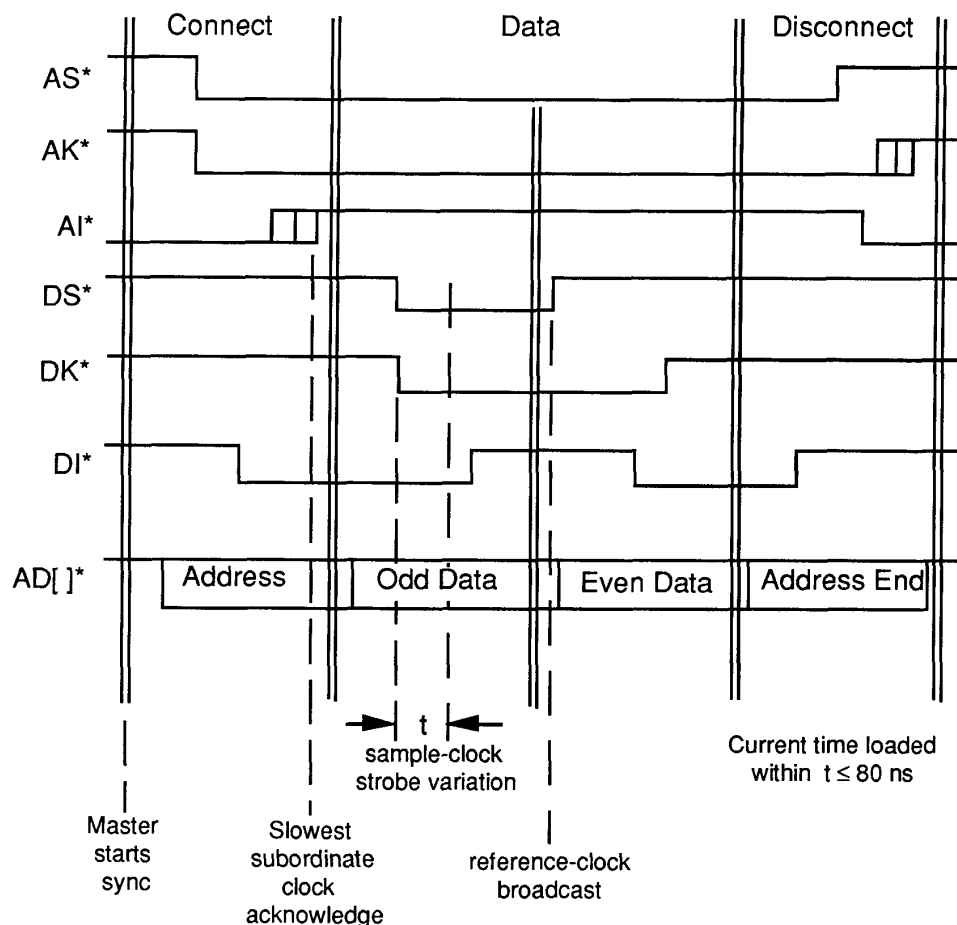


Fig 7
Clock Synchronization

3.1.3.5.7 System Clock Initialization. On system power-up, all clocks participating in clock synchronization must be initialized to a common initial reference time. The initial reference time may be an international standard time, some other time maintained locally, independent of node power or zero, depending on the application. The first step is for the reference clock node to update its own internal time_running counter with an initial reference time. How this is done is not specified.

Once the reference clock node time_running counter is initialized, this node may initialize all the other Futurebus+ clock nodes via the synchronization procedure previously described.

NOTE: It is up to the individual subordinate clock node's clock-manager process to distinguish between the initial reference time broadcast and all subsequent synchronization events, should such a distinction be necessary.

3.1.3.6 Unit Access. Futurebus+ nodes may contain a unit with resources that are designed to be accessible to other nodes on the bus. This unit may be made accessible in any of three address spaces:

- (1) Initial units space
- (2) Available address space
- (3) Indirect address space

A unit may be a memory, an I/O device, a bridge, a processor, or any other option.

3.1.3.6.1 Initial Units Space. *Initial units space* is the name given to the area between 2048 and 4095 in a node's CSR space. When 2 kbytes of addresses are enough for all units on the node, units' memory requirements may be mapped into this area. When units require additional address space, the MEMORY_BASE/MEMORY_BOUND and UNITS_BASE/UNITS_BOUND CSRs are used to specify additional memory accessible from other nodes.

3.1.3.6.2 Available Address Space. Unit architectures on a Futurebus+ node may have a need to make more than the 2048 bytes available in the initial units space of CSR space directly accessible on the bus. Available address space includes all locations not reserved for registers. When a node contains units with blocks of addresses to be mapped into available address space, the node will also contain ROM entries indicating the size and address alignment requirements of the address space.

Futurebus+ permits the identifications of two areas in the available address space. The Node ROM entries NODE_MEMORY_EXTENT (3.2.5.5.1) and NODE_UNITS_EXTENT (3.2.5.5.2) specify the extent of areas that can be accessed using Futurebus+ address. The starting address of an area is established when a system configuration process writes to the MEMORY_BASE or the UNITS_BASE CSR and the size of the allocated address space is set when the MEMORY_BOUND or the UNITS_BOUND CSR is written. There are no restrictions on the applications of either base/bounds pair.

A node's units address space cannot be placed on an arbitrary byte boundary. The MEMORY_BASE and UNITS_BASE have at least the 12 least significant bits equal to zero (a 4 kbyte boundary). Implementations may specify that UNITS_BASE has additional restrictions, e.g., the contents of UNITS_BASE may be restricted to values equal to 0 Mod 2^{20} without violating this specification. Specification of additional restrictions are defined in the NODE_MEMORY_EXTENT and NODE_UNITS_EXTENT ROM entries.

Access to node addresses mapped into available address space is controlled by the ENB bit in the UNITS_BASE or MEMORY_BASE CSRs. ENB is set to zero (access disabled) by Power-on, System Reset, and command_reset, in addition to being explicitly disabled by resetting the bit with a write to the UNITS_BASE or MEMORY_BASE CSRs.

In summary, access to the node's available address space is delimited by MEMORY_BASE and MEMORY_BOUND, UNITS_BASE and UNITS_BOUND, and is controlled by ENB bit.

3.1.3.6.3 Indirect Address Space. The INDIRECT_ADDRESS and INDIRECT_DATA CSRs may also provide access to node resources. Implementations may use these registers to access that part of node resources that has no other address window on the bus (e.g., unit-specific microcode or boot ROM). Software is expected to access data residing in the indirect address space by

- (1) Writing the indirect address space's address into the INDIRECT_ADDRESS CSR, and
- (2) Reading or writing the INDIRECT_DATA CSR to access the data at that address. This provides a flexible but relatively slow mechanism for indirectly accessing this data (which is expected to be accessed infrequently).

This standard does not require any node internal resources to be made available on the bus. This register pair does provide an indirect access mechanism to those who choose to use it.

Following a system or command_reset, INDIRECT_ADDRESS shall be set to zero. Bus Initialize does not change the contents of these CSRs. A read of the INDIRECT_ADDRESS CSR returns the last value written. The INDIRECT_ADDRESS and INDIRECT_DATA CSRs contain 32-bit values. INDIRECT_ADDRESS can address a maximum of 2^{30} locations.

3.2 Futurebus+ CSR Specification. This section specifies the CSRs to be used by most Futurebus+ nodes. These CSRs are divided into three classes:

- Core CSRs, described in 3.2.2.
- Futurebus+ dependent CSRs, described in 3.2.3.
- Node ROM entries, described in 3.2.4 and 3.2.5.

The initial units space may be defined by a vendor or specified in a unit-specific architecture standard. The following restrictions apply to the CSR address space identified in Table 5:

- The two least significant CSR address bits (**AD[1..0]***) are user-defined for general CSR accesses, and may be ignored, or may have user-specified effects. When referencing mailbox CSR addresses, the meaning of **AD[5..0]*** is specified in Section 9 of IEEE Std 896.1-1991 [2].
- Only an address width of 32 bits shall be supported.
- With the exception of WRITE_NO_ACKNOWLEDGE and ADDRESS_ONLY transactions to the message registers, “read unlocked” and “write unlocked” transactions shall be supported. Additional transaction types may also be supported.⁸
- Unimplemented fields and registers shall return zeros when read.
- Writes to read-only registers and to unimplemented fields and CSRs shall be ignored (no side affects).
- Block transfers to and from CSRs are permitted (but not required), as long as the serial sequence of operation is guaranteed to be maintained.
- Nodes that do not support block transfers to certain (or all) CSR addresses shall use **ED*** to terminate transactions after the first data beat.⁹
- When accessing core and Futurebus+ dependent CSRs, a data width of 32 bits shall be supported. Additional transaction widths and multiple data beats may also be supported.

When a CSR write is followed immediately by a CSR read, the second transaction may be split such that an operation or state change initiated by the first transaction will appear to have been immediate. As a last resort, CSR references may also be busied because some modules may not be able to respond immediately.

To complete the Futurebus+ definition, certain fields labeled reserved in IEEE Std 896.1-1991 [2] are assigned a specified function in this standard. Fields and addresses reserved in this specification shall not be used for vendor- or unit-defined functions. In cases of conflict or ambiguity, this standard is subordinate to IEEE Std 896.1-1991.

Futurebus+ nodes shall decode references to the local bus number (stored in the NODE_IDS CSR), and to the bus alias number (1023).

The supported addresses are specified in Table 1.

3.2.1 CSR Memory Space Specification. The CSR address space shall be allocated as shown in Table 5.

⁸ The target address of a split response transaction is CSR space.

⁹ This is specified in IEEE Std 896.1-1991 [2]. This is not an error case.

Table 5
CSR Address Space Allocation

AREA NAME	CSR ADDRESS RANGE
Core CSRs	0—508
Futurebus+ dependent CSRs	512—1020
ROM registers	1024—2044
Initial units space	2048—4092

3.2.2 Core CSRs. (See Table 6.) This section specifies the core CSRs required by Futurebus+ modules. It will be noted that the CSRs are not allocated sequentially. Registers omitted are reserved for the core CSR standard and shall not be defined by a bus or vendor standard. Byte offsets into the node address space are used throughout. The CSRs in this section shall support A32 and D32. CSRs may be accessed using any other data width supported by the node.

Table 6
Core CSRs and Addresses

REGISTER NAME	CSR ADDRESS
STATE_CLEAR	0
STATE_SET	4
NODE_IDS	8
RESET_START	12
INDIRECT_ADDRESS	16
INDIRECT_DATA	20
SPLIT_TIMEOUT	24, 28
ARGUMENT	32, 36
TEST_START	40
TEST_STATUS	44
UNITS_BASE	48, 52
UNITS_BOUND	56, 60
MEMORY_BASE	64, 68
MEMORY_BOUND	72, 76
INTERRUPT_TARGET	80
INTERRUPT_MASK	84
CLOCK_VALUE	88, 92
CLOCK_TICK_PERIOD	96, 100
CLOCK_STROBE_ARRIVED	104, 108
CLOCK_STROBE*	112
CLOCK_REFERENCE†	120, 124
MESSAGE_REQUEST	128,..., 188
MESSAGE_RESPONSE	192,..., 252
Reserved by IEEE Std 1212-1991 [12]	256,..., 380
ERROR_LOG_BUFFER	384,..., 508

* Same meaning as the clock unit CLOCK_INFO0 CSR specified in IEEE Std 1212-1991 [12].

† Same meaning as the clock unit CLOCK_INFO2 and CLOCK_INFO3 CSR specified in IEEE Std 1212-1991 [12].

3.2.2.1 STATE_CLEAR and STATE_SET CSRs. (See Table 7 and Fig 8.) The STATE_CLEAR and STATE_SET CSRs shall be used by a remote node to control and report state transitions. Writing ones to bits in the STATE_CLEAR CSR (address 0) shall cause the corresponding state bits to be cleared. Writing ones to bits in the STATE_SET CSR (address 4) shall cause the corresponding state bits to be set. Writing zeros to bits in these CSRs will have no effect on the corresponding state bits. Reading either the STATE_CLEAR or STATE_SET CSRs shall return the current contents of the internal state bits that make up these two CSRs.

Table 7
STATE_CLEAR and STATE_SET CSR Bit Descriptions

BIT FIELD	REQ OPT	ACCESS TYPE	POWER-UP	SYSTEM RESET	BUS INITIALIZE
Unit-Specific (16)	OPT				
The bits in this field shall be defined by the node's unit architectures.					
Bus-Dependent (8)	OPT				
The bits in this field shall be defined by individual profiles.					
ELOG	OPT	R/W	0	0	0
This bit shall be set when an error is detected in the node and the ERROR_LOG_BUFFER has been written. Operating system software is expected to clear the ELOG bit after saving the contents of the node's ERROR_LOG_BUFFER. This is the logical "OR" of specific bits in ERROR_HI and ERROR_LO.					
ATN	OPT*	R/W	0	0	no change
When this bit is set to a one, it shall cause the housekeeping operations performed prior to live removal to be initiated. The bit will remain a one and can be read while the housekeeping operation is being performed. When housekeeping operations are complete the module shall power down. Setting ATN in one node will cause the ATN bit in the other node on the same module, if present, to be set also.					
OFF	OPT	R/W	0	0	no change
Setting this bit shall cause power to be removed from the node's functional logic without waiting for the housekeeping operations to be performed. Thus, this bit will never be read as a one. Setting the OFF bit in one node will cause the OFF bit in the other node on the same module, if present, to be set also.					
STATE	OPT	R/W	Reflect State	Reflect State	Reflect State
State is a two-bit field that shall indicate whether the node is in the running, initializing, testing, or dead state. The states shall be interpreted as:					
Running 0 0 Initialization was successful, now running.					
Initializing 0 1 Initialization reset and test in progress. The TEST_STATUS CSR may contain additional information.					
Testing 1 0 Testing of the complete node is in progress. The node is unavailable for normal operation. The Testing state is entered only as a side-effect of a write to the TEST_START CSR that mandates a change of state. The TEST_STATUS CSR may contain additional information.					
Dead 1 1 A fatal error has been detected. The node is not operational. When this state is set, other information cannot be trusted.					

* Required if live insertion is implemented.

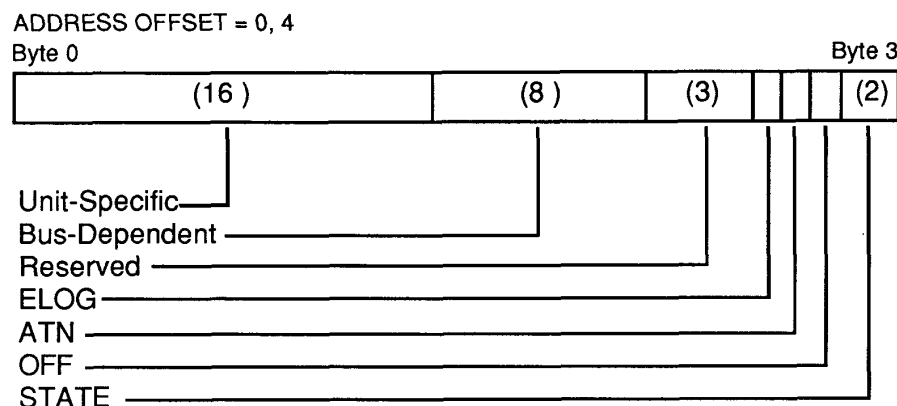


Fig 8

Fields in the STATE_CLEAR and STATE_SET CSRs Used by Futurebus+

3.2.2.2 NODE_IDS CSR. (See Table 8 and Fig 9.) The most significant 16-bits of the NODE_IDS CSR shall contain the node's identifier. If the node is designed to never use the distributed arbitration bus, then the least significant 16-bits shall be reserved (i.e., when read, these bits shall always return zeros). The least significant 8 bits of the NODE_IDS CSR may allow a system configuration process to set a default priority field in a target node for use in distributed arbitration and arbitrated messages. Specifically, the default priority may be used to set the PR[7..0] field during distributed arbitration and when sending arbitrated messages when no other priority value is explicitly provided by node-internal hardware or software. This is useful in controlling non-intelligent I/O modules in systems that employ distributed arbitration. If the node is designed to never use the distributed arbitration bus or if the node is designed to always provide priority values for PR[7..0], then the least significant 16 bits of the NODE_IDS CSR shall be reserved (i.e., when read, these bits shall always return zeros). The combination of GEOGRAPHICAL_ADDRESS and NODE_SIDE is called offset_id in IEEE Std 1212-1991 [12] and NODE_ADDRESS in this standard.

Note: The round robin bit shall not be controlled by the contents of this CSR.

Table 8
NODE_IDS CSR Bit Descriptions

BITS FIELD	REQ OPT	ACCESS TYPE	POWER-UP	SYSTEM RESET	BUS INITIALIZE
BUS_ADDRESS (10)	REQ	R/W	1023	1023	no change
It shall be possible for software to set the BUS_ADDRESS field to the current bus address. It is not expected that the BUS_ADDRESS will be changed while there are outstanding split transactions.					
GEOGRAPHICAL_ADDRESS (5)	REQ	RO	no change	no change	no change
GEOGRAPHICAL_ADDRESS shall reflect the geographical address signals applied to the GA[4..0]* lines at reset time. It shall not be modified by software or any form of reset. All nodes in a module shall have identical values in this field.					