

Information technology— Microprocessor systems— Futurebus+ —Logical protocol specification

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

Abstract: This International Standard provides a set of tools with which to implement a Futurebus+ architecture with performance and cost scalability over time, for multiple generations of single- and multiple-bus multiprocessor systems. Although this specification is principally intended for 64-bit address and data operation, a fully compatible 32-bit subset is provided, along with compatible extensions to support 128- and 256-bit data highways. Allocation of bus bandwidth to competing modules is provided by either a fast centralized arbiter, or a fully distributed, one or two pass, parallel contention arbiter. Bus allocation rules are provided to suit the needs of both real-time (priority based) and fairness (equal opportunity access based) configurations. Transmission of data over the multiplexed address/data highway is governed by one of two intercompatible transmission methods: a) a technology-independent, compelled-protocol, supporting broadcast, broadcast, and transfer intervention (the minimum requirement for all Futurebus+ systems), and b) a configurable transfer-rate, source-synchronized protocol supporting only block transfers and source-synchronized broadcast for systems requiring the highest possible performance. Futurebus+ takes its name from its goal of being capable of the highest possible transfer rate consistent with the technology available at the time modules are designed, while ensuring compatibility with all modules designed to this standard both before and after. The plus sign (+) refers to the extensible nature of the specification, and the hooks provided to allow further evolution to meet unanticipated needs of specific application architectures. It is intended that this International Standard be used as a key component of an approved IEEE Futurebus+ profile.

Keywords: bus architecture, Futurebus+, logical protocol, multiprocessor systems

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

Copyright © 1994 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1994. Printed in the United States of America.

ISBN 1-55937-373-3

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

April 27, 1994 SH16816

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 that have expressed an interest in participating in the development of the standard.

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

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

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its 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 may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers, Inc. does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user of such technology to obtain all necessary permissions.

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and nongovernmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

In 1993, ANSI/IEEE Std 896.1-1991, together with IEEE Std 896.1a-1993, *Errata, Corrections and Clarifications*, was adopted by ISO/IEC JTC 1, as draft International Standard ISO/IEC DIS 10857. This edition incorporates IEEE Std 896.1a-1993 into the text of ANSI/IEEE Std 896.1-1991.

Introduction

(This introduction is not a normative part of ISO/IEC 10857: 1994, but is included for information only.)

The following is a list of those who were members of the IEEE Futurebus+ Working Group at the time ANSI/IEEE Std 896.1-1991 was approved:

Paul L. Borrill, Chair

Barbara Aichinger	Ian Dobson	Subasis Laha
Ray Alderman	Emer Dooley	Cees Lambretche
Hamid Amirazzi	Sam Duncan	Dick Lawrence
Duane Anderson	Chris Eck	Mike Lazar
Harrison Beasley	Bill Evertz	Jim Leahy
Janos Biri	Wayne Fischer	Kent Leung
Martin Blake	Mike Foster	Joel Liblove
Richard Boberg	Joseph George	Thanos Mentzelopoulos
Andy Bonafini	Larry Gilbert	Klaus Müller
David Brash	Jim Goodman	Chris Nichols
David Brearly	Robert Greiner	Jim Nicholson
David Brewer	David Gustavson	Ronald Niederhagen
Marc Briel	Emil Hahn	Mira Pauker
Charles Brill	David Hartig	Chet Pawlowski
Jim Brown	David Hawley	Clarence Peckham
Mark Bunker	Lyre Hevle	Shlomo Pri-Tal
John Campbell	Billy Ho	Surinder Rai
Jay Cantrell	Mike Humphrey	Mike Raynham
Stephen Cecil	John Hyde	Jack Regula
Kim Clohessy	Ed Jacques	Bill Ruszczyk
Paul Cook	David James	Ali Sarabi
Dante Del-Corso	Greg Jewell	James Scaminaci
Ernie Crocker	Anatol Kaganovich	Dennis Schmitz
Jon Crowell	Hans Karlsson	Craig Scott
Steve Diess	David Kemp	Don Senzig
Paul Dixon	Ralph Lachenmaier	Lui Sha

Dan Sieworek
Mike Snodgrass
Michael Sweeney
Fahad Tabrizzi
Matthew Taub
Mike Teener
Judy Teske

Morton Thayer
John Theus
Mike Thompson
Nigel Topham
Mary Vernon
Harvey Walthersdorf
Eike Waltz

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

The following persons were on the balloting committee of ANSI/IEEE Std 896.1-1991:

William B. Adams
Sid Ahuja
Mohammad Al-Malki
John Allen
Richard P. Ames
Duane L. Anderson
Jack Arabian
R. V. Balakrishnan
David M. Barnum
Harrison A. Beasley
Janos Biri
Kyle M. Black
John Black
William P. Blase
Jack L. Blevins
David Brearley
Charles Brill
Lyle Burnett
Luis-Felipe Cabrera
Clyde Camp
Donald Chi
Kim Clohessy
David Cohen
Paul D. Cook
Robert Crowder
Jonathan C. Crowell
Philip D'Angelo
Ana Maria Dealvare
Stephen Deiss
Dante Del Corso
Su Dongzhuang
Mike Dorsett
Samuel H. Duncan
Sourav Dutta
Jeffrey S. Ebeling
William P. Evertz
Harry D. Feit
Wayne Fischer
Gordon Force
Andrew Fraser
Joseph D. George
Andy Glew
Patrick Gonias
Willard Graves

William Groseclose
David B. Gustavson
Thomas W. Harkaway
David Hawley
Herbert Hecht
Rick Henderson
Frank Horn
Scott Hopkinson
Zoltan R. Hunor
Peter J. Ilieve
Bob Jacobsen
Edgar Jacques
David V. James
Kenneth Jansen
Jack R. Johnson
Anatol Kaganovich
Hans Karlsson
David Keeney
Willis K. King
Hubert Kirrman
Ernst H. Kristiansen
Thomas M. Kurihara
Tuvia Lamdan
Glen Langdon
Thomas Leonard
Per Lindman
William Lindow
Rollins Linser
Wayne M. Loucks
Anthony G. Lubowe
Andy J. Luque
Roy Maurer
William McDonald
Darrell B. McIndoe
Brace Millard
Lee Minsuk
James M. Moidel
James Moloney
J.D. Nicoud
Tadahiko Nishimukai
Duane J. Northcutt
Gregory C. Novak
Michael Orlovsky
Jame R. Otto
Dick Palmer

Mira Pauker
Donald Pavlovich
James M. Pexa
Arthur V. Pohm
Bruno R. Preiss
Shlomo Pri-tal
Greg Prom
Richard Rawson
Michael Raynham
Ed Rodriguez
Tom Sakoda
Debabrata Sarma
Carl Schmiedekamp
Norman Schneidewind
Eugene C. Schramm
David Seraphin
Philip Shutt
Michael R. Sitzter
Michael Smolin
Benjamin Stoppe, Jr.
Paul Sweazey
Daniel Tabak
Darius Tanksalvala
Daniel Tarrant
Michael Teener
Michael G. Thompson
Carsten Thomsen
Joseph P. Trainor
Robert Tripi
Joseph G. Tront
Robert J. Voight
Eike Waltz
David R. Weller
Walter L. Whipple
Thomas Wicklund
Hans A. Wiggers
Mark Williams
John S. Willy
Andrew Wilson
John Wise
Joel Witt
David L. Wright
Qiufeng Wu
Oren Yuen

When the IEEE Standards Board approved ANSI/IEEE Std 896.1-1991 on September 26, 1991, it had the following membership:

Marco 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

The following is a list of those who were members of the IEEE Futurebus+ Working Group at the time IEEE Std 896.1a-1993 was approved:

Samuel H. Duncan, Chair

Harrison Beasley
Kim Burris
Jay Cantrell
Steve Cecil
Steve DiCamillo
R. Paul Dixon
Ian Dobson
Karl Franklin

Joseph D. George
Claes-Goran Gustavsson
Emil Hahn
Peter Izzo
Ed Jacques
Greg Jewell
Jim Leahy
Jeff Lear

Thanos Mentzelopoulos
Michael Munroe
Robert Schetlick
Gene Schramm
Richard Spratt
John Theus
Dean Van De Walker
Robert Widlicka

The following persons were on the balloting committee of IEEE Std 896.1a-1993:

Edward W. Aichinger
Ray S. Alderman
Richard P. Ames
Keith D. Anthony
Harrison A. Beasley
John Black
Charles Brill
Andrew J. Brough
Clyde Camp
Stephen J. Cecil
Andy Cheese
Kim Clohessy
Steven Cobb
David Cohen
Steven R. Corbesero
Ian Dobson
Jean-Jacques Dumont
Samuel Duncan
Christopher Eck

Wilhelm P. Evertz
Wayne Fischer
Gordon Force
Paul Fulton
Julio Gonzalez-Sanz
John Griffith
Michael C. Hayward
Edgar Jacques
Ralph Lachenmaier
Lak Ming Lam
Michael Lambrou
Karl E. McClure
Thanos Mentzelopoulos
Bruce Millard
Brian D. Morrison
Klaus Dieter Mueller
Elwood Parsons
Chandresh J. Patel

Steve Quinton
Michael L. Roby
Frederick E. Sauer
Robert Schetlick
Don Denzig
Patricia Smith
Joanne Spiller
Richard Spratt
Michael G. Thompson
Joseph P. Trainor
Robert Tripi
Yoshiaki Wakimura
Eike Waltz
Dave Wickliff
Robert Widlicka
Joel Witt
Mark Woodbury
David L. Wright
Yoshio Yamaguchi

When the IEEE Standards Board approved IEEE Std 896.1a-1993 on September 15, 1993, it had the following membership:

Wallace S. Read, *Chair*
Donald C. Loughry, *Vice Chair*
Andrew G. Salem, *Secretary*

Gilles A. Baril
Jose A. Berrios de la Paz
Clyde R. Camp
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ramiro Garcia
Donald N. Heirman

Jim Isaak
Ben C. Johnson
Walter J. Karplus
Lorraine C. Kevra
E.G. "Al" Kiener
Ivor N. Knight
Joseph L. Koepfinger*
D. N. "Jim" Logothetis

Don T. Michael*
Marco W. Migliaro
L. John Rankine
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp
Donald W. Zipse

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall

Richard B. Engelman

David E. Soffrin
Stanley I. Warshaw

IEEE Std 896.1-1991 was approved by the American National Standards Institute on April 28, 1992.

CLAUSE	PAGE
1. Overview	1
1.1 Scope	1
1.2 Normative references	3
2. Definitions and structure	3
2.1 Special word usage	3
2.2 Definitions	3
2.3 Signal conventions	7
2.4 Document structure	8
2.5 Futurebus+ logo	8
2.6 Bus line description	9
2.7 Attribute cross reference	12
2.8 Implementation mnemonics	21
3. Bus signaling environment	22
3.1 Description	22
3.2 Specification	22
4. Centralized arbitration	23
4.1 Description	23
4.2 Specification	25
5. Distributed arbitration and arbitrated messages	26
5.1 Description	26
5.2 Specification	41
6. Parallel protocol	51
6.1 Description	51
6.2 Specification	86
7. Bus/system management	115
7.1 Description	115
7.2 Specification	124
8. Cache coherence	131
8.1 Description	131
8.2 Specification	154
9. Message passing	159
9.1 Description	159
9.2 Specification	175
Annex A (informative) Bibliography	185

Information technology— Microprocessor systems— Futurebus+ —Logical protocol specification

1. Overview

1.1 Scope

This International Standard specifies the logical (relative timing and behavioral protocol) layer for a set of signal lines that constitute a multiple segment bus architecture, and for the interfacing of modules connected to a bus segment. This International Standard is intended to be used as a component within a profile (a collection of related specifications that must be used together by a product in order to claim conformance to a standard) to build systems with higher levels of compatibility.

Futurebus+ provides the means for the transfer of binary information between boards over one or more logical buses. Boards may contain any combination of one or more processors and local resources such as cache, memory, peripheral and communication controllers, etc. Figure 1 shows a block diagram of a typical application of Futurebus+.

Protocols are specified for the allocation of bus time to modules that need to conduct transactions with other modules over the bus. However, this International Standard does not mandate the priority rules for modules to use when competing for use of the bus. These are considered the privilege and responsibility of the system integrator. The International Standard includes a complete set of signaling rules to be followed by all modules in both the distributed and centralized control acquisition processes leading to bus mastership (clauses 4 and 5). The International Standard also gives a comprehensive set of signaling rules for all modules participating in a bus transaction (clause 6).

Most of the transfer protocols in this International Standard are compelled; that is, they are governed by a pure cause-and-effect relationship. This is what gives this International Standard its technology-independent nature. The compelled signaling provides a designer with a logical simplicity for what takes place in the protocols. As a result, there will be maximum compatibility between products designed to this International Standard throughout its operational lifetime.

With any bus, there is the dilemma of how much the standard should specify. There must be a balance between ensuring that all boards designed by a variety of manufacturers can operate together, while not restricting the users of the bus to any preconceived system design. Although the scope of this International Standard has been restricted to

exclude many of the system requirements associated with bus-based computer systems, these are being addressed in companion standards.

The common control and register interface to this series of standards for the Futurebus+, and to other proposed IEEE standards (in particular, IEEE Std 1596-1992 [B12]¹, IEEE P1014.1 [B2], and IEEE P1394 [B11]), is embodied in the unified CSR architecture standard, IEEE Std 1212-1991 [B7], along with a unified DMA architecture for moving data around a system without the need to pass through a processor (IEEE Std 1212.1-1993 [B8]).

This set of protocols has been designed to be as close to technology-independent as possible while maintaining a very high level of efficiency and performance. The bus signals may be implemented using any technology (TTL, Backplane Transceiver Logic, ECL, CMOS, GaAs, etc.) so long as the Futurebus+ signaling conditions are met (incident wave switching on the transmission-line signaling environment, along with the constraints on skew, crosstalk, and transmission reliability). However, in the interest of maximum compatibility between product families, implementations are expected to be associated with one or more IEEE Futurebus+ profiles, which specify the physical layer and set of transactions to suit a particular family of applications.

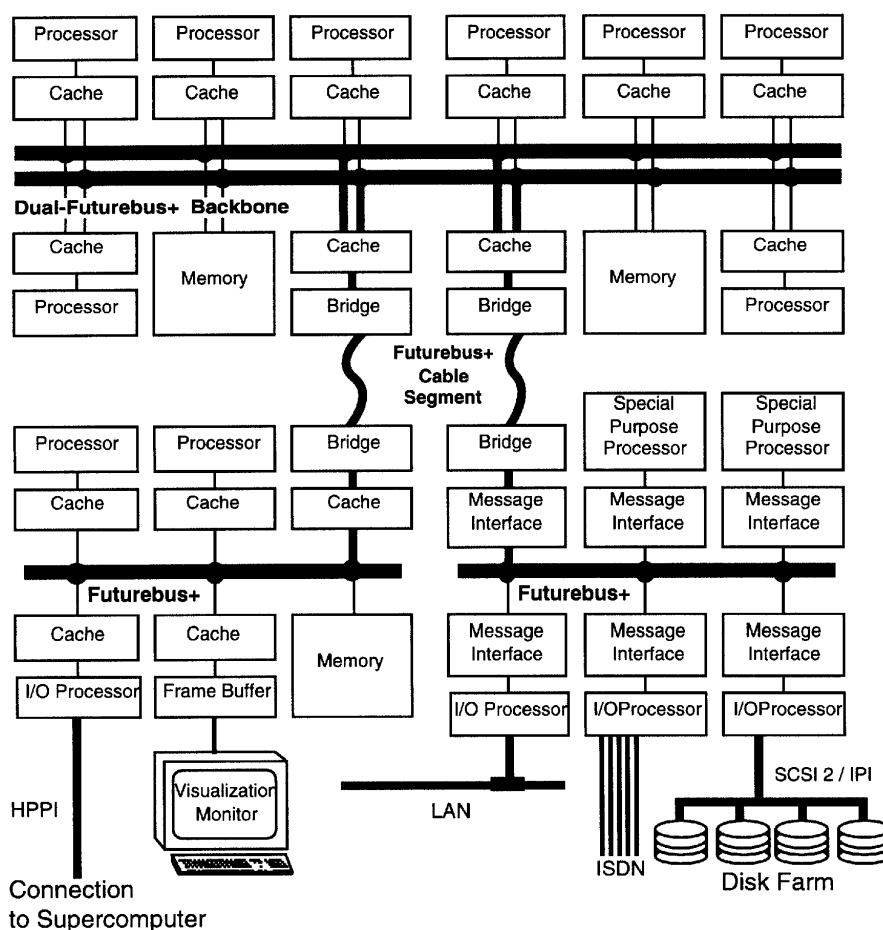


Figure 1— Interfaces in a family of typical Futurebus+ systems

¹The numbers in brackets correspond to those of the bibliography in annex A.

1.2 Normative references

The following standards contain provisions which, through references in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent edition of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

IEEE Std 896.2-1991, IEEE Standard for Futurebus+ — Physical Layer and Profile Specifications.²

IEEE Std 896.3-1993, IEEE Recommended Practices for Futurebus+.³

2. Definitions and structure

2.1 Special word usage

2.1.1 may: A keyword indicating flexibility of choice with no implied preference.

2.1.2 shall: A keyword indicating a mandatory requirement. Designers must implement all such mandatory requirements to ensure interoperability of ISO/IEC 10857 conformant products and claim conformance to this International Standard.

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

2.2 Definitions

2.2.1 activate: a) The action of applying a set of signals to a group of bus lines. b) The state of a group of bus lines when they carry signals.

2.2.2 address-only transaction: A bus transaction that does not include a data phase. The only information transferred is contained within the connection phase and, in some cases, the disconnection phase.

2.2.3 arbitration: The process of selecting the next bus master.

2.2.4 arbitrated message: A number broadcast on the arbitrated message bus lines to all modules on the bus.

2.2.5 assert: a) The action of applying a logic one signal to a bus line. b) The state of a bus line when the signal it carries represents a logic one.

2.2.6 * (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.

2.2.7 beat: An event that begins with the transition on a synchronization line by the master, followed by the release of an acknowledge line by one or more slaves. Command and data information may be transferred from the master to one or more slaves in the first half of the beat. During the second half of the beat the slaves may transfer capability, status, and data information back to the master.

2.2.8 block copy: A block copy operation is characterized by a long series of read or write transactions to sequential memory locations.

²IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hose Lane, P.O. Box 1331, Piscataway, Nj 08855-1331, USA.

³As this standard goes to press, IEEE Std 896.3-1993 is not yet published. It is, however, available in manuscript form from IEEE. Anticipated publication date is May 1994.

2.2.9 bus bridge: A bus bridge is an interconnect between two or more buses that 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).

2.2.10 bus line: The medium for the transmission of signals. Since Futurebus+ requires drivers with wireOR capability, a bus line may be driven by several modules simultaneously. Therefore, the signal carried by the bus line is the combination of signals applied to that line from each module.

2.2.11 bus tenure: The duration of a master's control of the bus; i.e., the time during which a module has the right to initiate and execute bus transactions.

2.2.12 bus transaction: An event initiated with a connection phase and terminated with a disconnection phase. Data may or may not be transferred during a bus transaction. See: **transaction**.

2.2.13 busy: If a slave is unable to accept a bus transaction from a master, it may issue a busy status to the master of the transaction. The master must relinquish the bus and may reacquire the bus and retry the transaction after a suitable time interval.

2.2.14 byte: A set of eight adjacent binary digits.

2.2.15 byte lane: A data path formed by eight data lines and one parity line and used to carry a single byte between system modules.

2.2.16 cache coherence: A system of caches is said to be coherent with respect to a cache line if each cache and main memory in the coherence domain observes all modifications of that same cache line. A modification is said to be observed by a cache when any subsequent read would return the newly written value.

2.2.17 cache memory: A buffer memory inserted between one or more processors and the bus, used to hold currently active copies of blocks from main memory. Cache memories exploit spatial locality by what is brought into a cache. Temporal locality is exploited by the strategy employed for determining what is removed from the cache.

2.2.18 coherence domain: A region in a multiple-cache system, inside of which, cache consistency measures are enforced. In a system that contains bus bridges, a coherence domain may or may not be extensible beyond the local bus through a bus bridge to remote buses.

2.2.19 coherence line: A data block for which cache consistency attributes are maintained.

2.2.20 compelled data transfer protocol: A technology-independent transfer mechanism in which the slave is compelled to provide a response before the master proceeds to the next transfer.

2.2.21 competitor: A module actively participating in the current control acquisition cycle of the arbitration process.

2.2.22 connected transaction: A transaction in which both the request and response are performed within the same bus transaction.

2.2.23 connection phase: A beat that begins with the assertion of the address synchronization line followed by the release of an address acknowledge line. It is used to broadcast the address and command information. Modules determine whether they wish to take part in the transaction based on this information.

2.2.24 control acquisition: The total of all bus activity associated with acquiring exclusive control of the bus.

2.2.25 copyback cache: A cache memory scheme with the attribute that data written from the processor is normally written to the cache rather than the main memory. Modified data in the cache is written to the main memory to avoid loss of the data when a cache line flush or replacement occurs.

2.2.26 CSR: Control and status register.

2.2.27 CSRA: Control and status register architecture. (See IEEE Std 1212-1991 [B7].)

2.2.28 data phase: A period within a transaction used to transfer data.

2.2.29 deadlock: A state that occurs when modules are awaiting actions that can only be performed by those waiting, and those waiting cannot perform the actions.

2.2.30 disconnection phase: A period within a transaction used to return the bus signals to their quiescent state. In addition, this phase might be used to transfer additional information required to perform or abort the requested operation.

2.2.31 doublet: A set of two adjacent bytes.

2.2.32 entrant: A live inserted module in the process of aligning itself with the arbitration protocol.

2.2.33 forward progress: A state in which a module is not blocked from performing the tasks necessary to achieve its goal. Forward progress is guaranteed only in the absence of deadlock or starvation.

2.2.34 geographical address: A unique identifier assigned to each physical module slot on the bus and assumed by any module connected to that slot.

2.2.35 global identification: A unique identifier assigned to each physical module slot in a system. This identifier would typically include both a bus identifier and a slot identifier. IEEE Std 1212-1991 [B7] specifies the format for such a global identifier.

2.2.36 intervening slave: The participating slave that, although not the repository of last resort of the requested data, finds it necessary to prevent the repository of last resort from providing the requested data. Having done so, the intervening slave provides the data instead.

2.2.37 livelock: A metastable situation in which some modules acquire and release resources in such a way that none of them makes forward progress.

2.2.38 locking: A facility whereby a module is requested to guarantee exclusive access to addressed data, blocking other modules from accessing that data. This allows indivisible operations to be performed on addressed resources.

2.2.39 master: A module that has acquired control of the bus through the control acquisition procedure.

2.2.40 master elect: A module that has won the most recent arbitration competition.

2.2.41 module: A collection of circuitry designed to perform specific functions that include an interface to Futurebus+.

2.2.42 monarch processor: The processor selected to manage the configuration and initialization of all modules on one logical bus. *Also:* monarch. An *emperor processor* is the monarch processor selected to direct the configuration and initialization of an entire system with multiple interconnected logical buses.

2.2.43 octlet: A set of eight adjacent bytes.

2.2.44 packet data transfer protocol: A very fast but technology-dependent noncompelled transfer mechanism that uses a compelled protocol over the entire packet to provide flow control.

2.2.45 parallel contention arbitration: A process whereby modules assert their unique arbitration number on a parallel bus and release signals according to an algorithm such that after a period of time the winner's number appears on the bus.

2.2.46 participating slave: A slave involved in a transaction as a selected slave, an intervening slave, a broadcast slave, or a slave involved in multiple packet mode.

2.2.47 preemption: The release of the bus by the current bus master due to the request of another module.

NOTE — In some systems, any module may cause preemption; in others, only a module with a higher priority request may cause preemption.

2.2.48 quadlet: A set of four adjacent bytes.

2.2.49 release: a) The action of applying a logic zero signal to a bus line. b) The state of a bus line when the signal it carries represents a logic zero.

2.2.50 repository of last resort: In a hierarchical memory (or cache-based) environment, a storage location that “owns” the only, or last remaining, copy of shatable data.

NOTE — It may be a unique source, an ultimate destination, or simply a “safe” repository of data that may not be invalidated, unless action is taken to preserve a copy of that data at some higher level in the memory (or cache) hierarchy. In a cache-only Futurebus+ system (e.g., one where even the main DRAM storage is also designed as a hardware cache), the repository of last resort begins life as the binding of an address to a physical location in one of the caches, along with the creation of the data by initialization, a copy from some higher level in the memory hierarchy, or by its arrival from some I/O device. This data may migrate around the system, and be owned by different caches at different times, provided no less than one copy of that data is maintained somewhere. A repository of last resort may end its life by an explicit instruction to “destroy” the data by migration to a higher level in the memory (or cache hierarchy), or by transfer of ownership through some I/O device to another system, storage device, or display.

2.2.51 request: A command, generated by a requester, to initiate an action on a responder. For a processor-to-memory read transaction, for example, the request transfers the memory address and command from the processor to memory. In the case of a split transaction, the request would be a separate bus transaction. In the case of a connected transaction, the request would be the connection phase of a bus transaction.

2.2.52 requester: A module that initiates a transaction by sending a request (containing address, command, and sometimes data).

2.2.53 responder: A module that completes a transaction by sending a response (containing the completion status and sometimes data).

2.2.54 response: A reply, generated by a responder, to complete a transaction initiated by a requester. For a processor-to-memory read transaction, for example, the response returns the data and status from the memory to the processor. In the case of a split transaction, the response would be a separate bus transaction. In the case of a connected transaction, the response would be the data and disconnection phases of a bus transaction.

2.2.55 round robin: A bus allocation rule where, after a module acquires and uses the bus, it will not be granted use of the bus again until all other modules currently requesting the bus at the same priority level have had control of the bus.

2.2.56 selected slave: A slave is selected during the connection phase by the master when it recognizes its address on the bus lines.

2.2.57 shared memory: The address space in the system accessible to all caching modules.

2.2.58 signal names: Where a group of bus lines are represented by the same letters, the lines within the group are numbered; e.g., **ADO***, **ADI***, **AD2***, etc. In order to represent a group of lines or signals in more convenient form, notation such as **AD[63..0]*** is used. Also, in these examples, the notation **AD[]*** is used to refer to all of the lines within the group.

2.2.59 skew: The difference between the propagation delays of two or more signals on any bus line.

2.2.60 slave: A module that can be addressed and is capable of participating in bus transactions.

2.2.61 snarf: The action taken by a module when it takes a copy of data passing by on the bus, even though it did not request it.

2.2.62 snoop: The action taken by a module on a transaction when it is not the master that originated the transaction or the repository of last resort for the data, but it still monitors the transaction. Cache memories snoop transactions to maintain coherence.

2.2.63 spatial locality: The tendency for a program to reference closely related clusters of memory addresses over short time intervals.

2.2.64 split transaction: A system transaction in which the request is transmitted in one bus transaction and the response is transmitted in a separate subsequent bus transaction.

2.2.65 starvation: A system condition that occurs when one or more module(s) perform(s) no useful work for an indefinite period of time due to lack of access to the bus or other system resources.

2.2.66 strong sequential consistency: A state exhibited by a system when each participating cache in the system observes all modifications to lines within itself in the same order as all other participating caches in the system. *See also: weak sequential consistency.*

NOTE — Futurebus+ allows modules to issue transactions that dynamically choose between following a weakly consistent behavior model (which implies greater concurrency and hence higher performance) and strongly consistent behavior models (which may be necessary to ensure correct operation of an algorithm written by a programmer not cognizant of the concurrency possible in different parts of a system).

2.2.67 system transaction: A complete operation, such as a memory read or write, as viewed from the initiating unit. A system transaction can be translated into one or more bus transactions by the Futurebus+ interface to complete the operation.

2.2.68 temporal locality: The tendency for a program to reference the same memory locations over short time intervals.

2.2.69 transaction: An event initiated with a connection phase and terminated with a disconnection phase. Data may or may not be transferred during a transaction. Often used instead of the more precise phrase “bus transaction” for the sake of brevity. *See: bus transaction; system transaction.*

2.2.70 unselected slave: A slave that does not recognize its address on the bus lines during the connection phase of a bus transaction.

2.2.71 weak sequential consistency: A state exhibited by a system when references to global synchronizing variables exhibit strong sequential consistency, and if no reference to a synchronizing variable is issued by any processor until all previous modifications to global data have been observed by all caches, and if no reference to global data is issued by any processor until all previous modifications to synchronizing variables have been observed by all caches. *See also: strong sequential consistency.*

2.3 Signal conventions

Figure 2 shows an example of the signal conventions used in this International Standard. The example is shown with open collector technology (although this International Standard is independent of this logic family). The signal that a specific 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 wire-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 a wire-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.

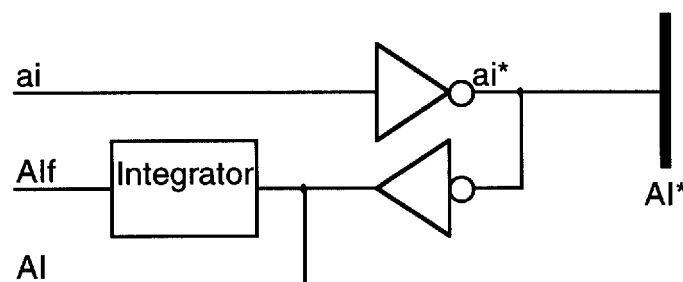


Figure 2— Signal conventions

2.4 Document structure

Each of clauses 3 through 9 is divided into two subclauses: one containing descriptions and one containing specifications. The numbers of the description subclauses begin with the clause number followed by “.1”. The numbers of the specification subclauses begin with the clause number followed by “.2”. For example, the description subclause of clause 3 is numbered 3.1, and the specification subclause is numbered 3.2.

The description subclauses are intended to help designers understand the operation of the bus and are *not intended to imply any requirements on equipment* built to this International Standard. The specification subclauses contain *all the requirements* with which equipment designed to this International Standard shall comply.

The specification subclauses are written using attributes (including CSR bits), states, and equations. This format, although somewhat difficult to read, allows the specification to be precise. Designers may wish to read all of the description subclauses prior to reading the specification subclauses.

Attributes are identified by uppercase with the underscore character separating words. For example, COMPELLED_STATUS, MASTER, and MASTER_ELECT are attributes. The symbols used in equations in order of precedence are as follows:

~	Negation
&	Logical AND
	Logical OR

For example, the specification “A COMPETITOR shall assert **ab6*** when **cn6** & COMPETE & (**cn7** | ~**AB7***)” would mean: A module with the COMPETITOR attribute set shall assert the signal **ab6*** when the **cn6** signal is set and the COMPETE attribute is set and either **cn7** is set or **AB7*** is released.

Attributes are logical entities that do not necessarily correspond to circuitry on a module. A machine's internal state variables may or may not conform to the attributes described in this International Standard. Any machine whose external behavior is equivalent to the specified machine conforms to this International Standard.

Although this International Standard indicates, where appropriate, possible areas where a designer may encounter metastability (the misbehavior of digital logic at the boundary between two synchronization domains), no guarantee is given that those warnings cover all possible causes of metastability; avoiding metastability must ultimately be the responsibility of the designer, since different application architectures require different synchronization boundaries.

2.5 Futurebus+ logo

The IEEE 896 logo shown in figure 3 is copyrighted by the IEEE Computer Society and may be used only on products that fully conform to all mandatory requirements of an IEEE-approved 896 profile document, or adhere to either this document (ISO/IEC 10857 :1993) or to earlier editions, IEEE Std 896.1-1991 or IEEE Std 896.1-1987.



Figure 3— IEEE 896 logo

2.6 Bus line description

Table 1 shows the list of bus lines, while 2.6.1.1–2.6.1.9 give a brief description of their function. The signals that use these lines are more fully described in later clauses.

Table 1— List of bus lines

Data Path Width		32-bit	64-bit	128-bit	256-bit
Information lines					
AD[63..0]*	Address/Data	32	64	64	64
D[255..64]*	Data	—	—	64	192
BP[31..0]*	Bus Parity	4	8	16	32
TG[7..0]*	TaG	8	8	8	8
TP*	Tag Parity	1	1	1	1
CM[7..0]*	CoMmand	8	8	8	8
CP*	Command Parity	1	1	1	1
ST[7..0]*	STatus	8	8	8	8
CA[2..0]*	CApability	3	3	3	3
Synchronization lines					
AS*, AK*, AI*	Address handshake	3	3	3	3
DS*, DK*, DI*	Data handshake	3	3	3	3
ET*	End of Tenure	1	1	1	1
Arbitrated message lines					
AB[7..0]*	Arbitrated Message Bus	8	8	8	8
ABP*	Arbitrated Message Bus Parity	1	1	1	1
AP*, AQ*, AR*	Arbitrated Message Synchronization	3	3	3	3
AC[1..0]*	Arbitrated Message Condition	2	2	2	2
Reset/bus initialize line					
RE*	REset	1	1	1	1
Central arbitration					
RQ[1..0]*	ReQuest	2	2	2	2
GR*	GRant	1	1	1	1
PE*	PreEmption	1	1	1	1
GA[4..0]*	Geographical Address	5	5	5	5
Total active signals		96	132	204	348*

* NOTE — For the number of 0V logic lines, serial bus lines, and other profile-related information, refer to IEEE Std 896.2-1991.

2.6.1 Information lines

The following bus lines are used during the execution of bus transactions.

2.6.1.1 AD[63..0]* address/data/lane deselect/request address/return address

These multiplexed bidirectional address/data lines carry the address during an address transfer. The address is always driven by the master and received by one or more slaves. During the data transfer these lines carry data. Data is driven by the master during write transactions, and by the selected or intervening slave during read transactions. During the first data beat of partial transfers **AD[31..0]*** carry lane deselect signals. During ing cached split transactions

AD[31..0]* carry the requesting module's address and priority and the responding module's status. During noncached split transactions **AD[31..0]*** carry the requesting module's global identification and transaction identification.

2.6.1.2 D[255..64]* data lines

These bidirectional data lines carry the data during a data transfer. Data is driven by the master during write transactions and the response portion of split read transactions. Data is driven by the selected or intervening slave during read transactions.

2.6.1.3 CM[7..0]* command

A set of lines that carry command information from the master to one or more slaves for the current transaction.

2.6.1.4 CP* command parity

The line that carries the parity for the command lines.

2.6.1.5 ST[7..0]* status

A set of lines that are activated by slaves to respond to the master's command. **ST[7..0]*** provide information both about the status of slaves and their disposition in regards to the requested transfer.

2.6.1.6 CA[2..0]* capability

A set of lines that a module activates to declare its ability to accommodate major modes of bus transactions. **CA[2..0]*** provide the basic mechanism that allows modules with various transaction capabilities to coexist in the system; e.g., a module that can respond to packet data transfers and a module that can only execute compelled data transfers.

2.6.1.7 BP[31..0]* bus parity

These lines carry parity for the address/data lines. There is one parity line for every 8 address/data lines.

2.6.1.8 TG[7..0]* tag

These bidirectional lines carry additional information related to the address/data lines. This International Standard does not prescribe the use of **TG[7..0]***; the system designer may use them to fit the specific requirements of the application. The number of tag lines active is specified in a control and status register.

2.6.1.9 TP* tag parity

The line that carries the parity code for **TG[7..0]***.

2.6.2 Synchronization lines

The synchronization lines coordinate the exchange of the address, command, capability, status, and data during a transaction.

2.6.2.1 AS* address synchronization

A synchronization signal that is asserted during the connection phase by the master to inform the system's slaves that the address and command information are valid. **AS*** is released during the disconnection phase to indicate that the disconnection data and command information are valid.

2.6.2.2 AK* address acknowledge

An acknowledge signal released by all modules during the disconnection phase to indicate to the master that their status information is valid.

2.6.2.3 AI* address acknowledge inverse

An acknowledge signal released by all modules during the connection phase to indicate to the master that their status and capability information is valid.

2.6.2.4 DS* data synchronization

A synchronization signal asserted or released by the master during the data transfer phase to indicate to the participating slaves that command information and write data are valid or that the master is ready to receive read data.

2.6.2.5 DK* data acknowledge

An acknowledge signal released by the participating slaves during the even beats or even packets of the data phase to indicate to the master that status information and read data are valid or write data has been received.

2.6.2.6 DI* data acknowledge inverse

An acknowledge signal released by the participating slaves during odd beats or odd packets of the data phase to indicate to the master that status information and read data are valid or write data has been received.

2.6.2.7 ET* end of tenure

This signal marks the end of tenure of the current master. The release of this signal notifies the master elect that it is the new master.

2.6.3 Distributed arbitration and arbitrated message lines

2.6.3.1 AB[7..0]* arbitration bus

A set of lines that carry a number that signifies the precedence of competitors during the arbitrated message process and the distributed arbitration process.

2.6.3.2 ABP* arbitration parity

ABP* carries the parity for AB[7..0]*.

2.6.3.3 AP*, AQ*, AR* control acquisition synchronization

Handshake lines that perform a cyclic handshake sequence that controls the sequencing of the arbitrated message process and the distributed arbitration process.

2.6.3.4 AC[1..0]* arbitration condition

Lines that control the arbitrated message process and the distributed arbitration process.

2.6.4 RE* reset/bus initialize

A line asserted by a module to initialize the bus interface logic of all modules to a predefined state, and provide a reset signal for use on the boards. The duration of the asserted state on **RE*** is used to decide whether the signal represents live insertion, bus initialize, or system reset.

2.6.5 Central arbiter

In systems implementing a central arbiter, these signals are used for bus master selection as described in the following subclauses.

2.6.5.1 PE* preemption

The central arbiter asserts this line to indicate to the current master that a preemptive condition exists and the master should relinquish bus mastership.

2.6.5.2 GR* grant

This line is asserted by a central arbiter to grant bus mastership. There is a separate grant line from the central arbiter to each module on the bus.

2.6.5.3 RQ[1..0]* request

One or both of these lines are asserted by modules to request bus mastership. There are two separate request lines from each module on the bus to the central arbiter.

2.6.6 GA[4..0]* geographical address

A set of static signals from the backplane that encode the slot number of the module.

2.7 Attribute cross reference

Each attribute lists the clauses that refer to it. A clause in **boldface type** includes the definition of the attribute.

32_BIT_ADDRESSES_CAPABLE **7.2.4.1.1**

32_BIT_CAPABLE **6.2.1.3**

32_BIT_DATA_CAPABLE **7.2.4.1.1**

64_BIT_ADDR **6.2.1.3**

64_BIT_DATA **6.2.1.3**, 6.2.2.5.1, 6.2.2.5.3

64BIT_ADDRESS_CAPABLE 6.2.1.3, **7.2.4.1.1**

64BIT_DATA_CAPABLE **7.2.4.1.1**

128_BIT_DATA **6.2.1.3**, 6.2.2.5.1, 6.2.2.5.2, 6.2.2.5.3

128BIT_DATA_CAPABLE **7.2.4.1.1**

256_BIT_DATA **6.2.1.3**, 6.2.2.5.1, 6.2.2.5.2, 6.2.2.5.3

256BIT_DATA_CAPABLE 7.2.4.1.1

ADDR_DECODED 6.2.1.7.2, **6.2.1.8**, 6.2.2.1.3, 6.2.3.2, 8.2.2, 9.2.1

ADDR_WIDTH **6.2.1.3**, 6.2.2.2.8, 6.2.2.5.1, 6.2.2.5.3

ADDRESS_ONLY **6.2.1.4**, 8.2.3

ADDRESS_ONLY_LOCKED **6.2.1.4**, 6.2.1.5, 6.2.1.8, 6.2.2.5.1

ADDRESS_ONLY_UNLOCKED **6.2.1.4**, 6.2.1.8, 6.2.2.5.1

ADDRESS_PARITY_ERROR **6.2.1.7.2**, 6.2.1.8

ADDRESS_TAG_PARITY_ERROR **6.2.1.7.2**

ARBITRATED_MESSAGE_REQUEST **5.2.1**, 5.2.4, 5.2.7.1.1, 5.2.9.1, 5.2.9.2, 5.2.9.6, 7.2.4.2.1

ARBITRATED_MESSAGE_SENT **5.2.1**, 5.2.9.1, 5.2.9.2, 5.2.9.6

ARBITRATION_COMPARISON_ERROR **5.2.6**

ARBITRATION_ERROR 5.2.4, **5.2.6**, 5.2.7.2.1, 5.2.8.1, 5.2.8.3–5.2.8.5, 5.2.9.1, 5.2.9.35.2.8.5–5.2.9.5

ARBITRATION_OWNER **5.2.3**, 5.2.4, 5.2.7.1.3, 5.2.8.1, 5.2.8.2, 5.2.8.5, 5.2.8.6

ARBITRATION_PARITY_ERROR **5.2.6**, 5.2.8.1, 5.2.8.4, 5.2.9.1, 5.2.9.4

ARBITRATION_TIMEOUT **5.2.6**, 5.2.7.1.2, 5.2.7.1.3, 5.2.8.1, 5.2.8.3, 5.2.8.5, 5.2.9.1, 5.2.9.3, 5.2.9.5

ASSERT_CS 6.2.2.4.5, **8.2.2**

ATTRIBUTES_QUEUED **8.2.2**

BEAT_ERROR **6.2.1.7.2**, 6.2.2.4.7

BEAT_ERROR_STATUS **6.2.1.7.1**, 6.2.1.8

BROADCAST_CALL_ADDRESS **6.2.1.7.2**, 6.2.2.4.4

BROADCAST **6.2.1.7.2**, 6.2.1.8, 6.2.2.1.5, 6.2.1.8, 6.2.2.4.4, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, 6.2.3.14, **9.2.1**

BROADCAST_SELECT_ATTRIBUTE **9.2.1**

BROADCAST_STATUS **6.2.1.7.1**, 6.2.1.8, 6.2.2.1.4, 6.2.2.1.5, 6.2.2.1.6, 6.2.2.5.1–6.2.2.5.3, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, 6.2.3.22

BUS_HOLD 5.2.1–5.2.3, 6.2.1.8, 6.2.2.1.1, **7.2.1**

BUS_IDLE **7.2.1**

BUS_IDLE_IUS 5.2.7.1.3, 6.2.2.1.3, **7.2.1**, 7.2.2, 7.2.3.4

BUS_INIT **7.2.1**, 7.2.3.3, 7.2.4.2.1–7.2.4.2.6

BUSY **6.2.1.7.2**, 6.2.2.4.2, 8.2.2

BUSY_STATUS 5.2.3, **6.2.1.7.1**, 6.2.1.8

CACHE **8.2.1**, 8.2.3

CACHE_AGENT **8.2.1**, 8.2.2, 8.2.3, 8.2.5

CACHE_CAPABLE **7.2.4.1.1**

CACHE_CMD **6.2.1.4**, 6.2.2.4.5

CACHE_OP_CMPLT **8.2.2**

CACHED **8.2.2**, 8.2.3–8.2.5

CANCEL **5.2.4**, 5.2.7.2.2, 5.2.8.1, 5.2.8.4, 5.2.8.5, 5.2.9.1, 5.2.9.4

CAPABILITY_ERROR **6.2.1.7.2**

CENTRAL_ARBITER 4.2.2.1, 4.2.2.2, 5.2.1–5.2.3, 5.2.7.1.1, 5.2.7.1.3, **7.2.4.2.1**

CENTRAL_MESSAGE_REQUEST **5.2.1**, 5.2.4, 5.2.7.1.1, 5.2.9.1, 5.2.9.2, 5.2.9.6, 7.2.4.2.1

CENTRAL_MESSAGE_SENT **5.2.1**, 5.2.9.1, 5.2.9.2, 5.2.9.6

CENTRALIZED_MASTER **4.2.1**, 4.2.2.3, 6.2.1.1

COMPARE_SWAP **6.2.1.5**, 6.2.1.8, 6.2.3.3–6.2.3.8, 7.2.4.1.1

COMPARE_SWAP_CAPABLE **7.2.4.1.1**

COMPELLED **6.2.1.1**, 6.2.2.3.2

COMPELLED_STATUS **6.2.1.1**, 6.2.1.6, 6.2.1.7.2, 6.2.1.8, 6.2.2.2.1, 6.2.3.1, 6.2.3.2, 6.2.3.22, 6.2.3.23

COMPETE **5.2.5**, 5.2.7.3.1–5.2.7.3.9, 5.2.8.2, 5.2.8.5, 5.2.9.2, 5.2.9.5

COMPETITOR **5.2.4**, 5.2.5, 5.2.6, 5.2.7.3.1–5.2.7.3.9, 5.2.8.2–5.2.8.6, 5.2.9.2–5.2.9.6

CONNECTION_COMMAND_PARITY_ERROR **6.2.1.7.2**, 6.2.1.8

CONNECTION_INFO_CAPTURED 6.2.1.7.1, **6.2.1.8**, 6.2.2.1.1

CONNECTION_PHASE 6.2.1.1–6.2.1.4, 6.2.1.7.2, **6.2.1.8**, 6.2.2.1.1, 6.2.2.1.6, 6.2.2.2.1–6.2.2.2.8, 6.2.2.3.1, 6.2.2.5.1, 6.2.2.5.3

CONNECTION_STATUS 6.2.1.1, 6.2.1.2, **6.2.1.8**, 6.2.2.4.1, 6.2.3.2

COPY_BACK **6.2.1.4**, 6.2.1.8, 6.2.2.5.1, 6.2.3.1, 8.2.2–8.2.4

DATA_COMMAND_PARITY_ERROR **6.2.1.7.2**

DATA_INFO_CAPTURED 6.2.1.1, 6.2.1.4, 6.2.1.7.1, **6.2.1.8**, 6.2.2.1.1, 6.2.2.1.4–6.2.2.1.6, 6.2.3.3, 6.2.3.5, 6.2.3.7, 6.2.3.9, 6.2.3.11, 6.2.3.13, 6.2.3.15, 6.2.3.17

DATA_LENGTH_0 **6.2.1.6**, 6.2.2.2.6

DATA_LENGTH_1 **6.2.1.6**, 6.2.2.2.7

DATA_LENGTH_2 6.2.1.6, 6.2.2.2.8

DATA_PARITY_ERROR 6.2.1.7.2

DATA_PHASE 6.2.1.1, 6.2.1.4–6.2.1.6, 6.2.1.7.2, 6.2.1.8, 6.2.2.1.4, 6.2.2.2.1–6.2.2.2.3, 6.2.2.2.5–6.2.2.2.8, 6.2.2.5.1–6.2.2.5.3, 6.2.3.2, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, 6.2.3.16, 6.2.3.18

DATA_TAG_PARITY_ERROR 6.2.1.7.2

DATA_WIDTH_0 6.2.1.3, 6.2.1.8, 6.2.2.2.6, 6.2.2.5.1, 6.2.2.5.2, 6.2.2.5.3

DATA_WIDTH_I 6.2.1.3, 6.2.1.8, 6.2.2.2.7, 6.2.2.5.1, 6.2.2.5.2, 6.2.2.5.3

DEPOSITION **5.2.3**, 5.2.4, 5.2.8.1, 5.2.8.4

DISC_INFO_CAPTURED **6.2.1.8**, 6.2.2.1.2, 8.2.2

DISCONNECTION_COMMAND_PARITY_ERROR **6.2.1.7.2**

DISCONNECTION_DATA_PARITY_ERROR **6.2.1.7.2**

DISCONNECTION_END 6.2.1.7.2, **6.2.1.8**, 6.2.2.3.1–6.2.2.3.3, 6.2.2.4.2–6.2.2.4.8, 6.2.3.23, 7.2.1

DISCONNECTION_PHASE 6.2.1.4–6.2.1.6, **6.2.1.8**, 6.2.2.2.1, 6.2.2.2.2, 6.2.2.2.5–6.2.2.2.8, 6.2.2.5.1, 6.2.2.5.3, 6.2.3.2, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, 6.2.3.16, 6.2.3.18, 6.2.3.22

DISCONNECTION_TAG_PARITY_ERROR **6.2.1.7.2**

DISTRIBUTED_ARBITER_REQUEST 5.2.2, **5.2.3**, 5.2.4, 5.2.7.1.1, 5.2.8.1, 5.2.8.2, 5.2.8.6, 7.2.4.2.2

DISTRIBUTED_GRANT **5.2.3**

DISTRIBUTED_MASTER **5.2.3**, 5.2.8.1, 5.2.8.6, 6.2.1.1

DISTRIBUTED_MESSAGE **5.2.2**, 5.2.3, 5.2.4, 5.2.8.1, 5.2.8.4

DISTRIBUTED_MESSAGE_ENABLE 5.2.1, 5.2.2, **7.2.4.2.1**

DISTRIBUTED_MESSAGE_REQUEST **5.2.2**, 5.2.3, 5.2.4, 5.2.7.1.1, 5.2.8.1, 5.2.8.2, 5.2.8.6, 7.2.4.2.1

DISTRIBUTED_MESSAGE_SENT **5.2.2**, 5.2.8.1, 5.2.8.2, 5.2.8.6

END_OF_DATA 6.2.1.7.1, **6.2.1.7.2**, 6.2.2.4.2

END_OF_DATA_STATUS **6.2.1.7.1**, 6.2.1.8

ENTRANT **7.2.1**, 7.2.2

EXCLUSIVE_MODIFIED 8.2.2, **8.2.3**

EXCLUSIVE_MODIFIED_STATUS **8.2.2**, 8.2.5

EXCLUSIVE_UNMODIFIED 8.2.2, **8.2.3**

EXCLUSIVE_UNMODIFIED_STATUS **8.2.2**

FETCH_ADD_CAPABLE **7.2.4.1.1**

FETCH_ADD_BIG **6.2.1.5**, 6.2.1.8, 6.2.3.3–6.2.3.6, 7.2.4.1.1

FETCH_ADD_LITTLE **6.2.1.5**, 6.2.1.8, 6.2.3.3–6.2.3.6, 7.2.4.1.1

HIGH_SPEED_ENABLE 6.2.1.1, **7.2.4.2.2**

ILLEGAL_OPERATION_ERROR **6.2.1.7.2**

INIT 4.2.1, 4.2.2.1, 4.2.2.2, 5.2.1–5.2.5, 5.2.7.1.1–5.2.7.1.3, 6.2.1.1–6.2.1.6, 6.2.1.7.1, 6.2.1.7.2, 6.2.1.8, 6.2.2.1.1–6.2.2.1.6, 6.2.2.2.1–6.2.2.2.8, 6.2.2.3.1–6.2.2.3.3, 6.2.2.4.1–6.2.2.4.8, 6.2.2.5.1–6.2.2.5.3, 6.2.2.5.6, **7.2.1**, 9.2.1

INTERVENTION 6.2.1.7.2, 6.2.2.4.6, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, 6.2.3.14, 6.2.3.16, 6.2.3.18, **8.2.2**

INTERVENTION_STATUS **6.2.1.7.1**, 6.2.2.2.5

INVALID 8.2.2, **8.2.3**

INVALID_STATUS **8.2.2**, 8.2.3, 8.2.5

INVALIDATE **6.2.1.4**, 6.2.1.8, 6.2.2.5.1, 8.2.26.2.1.8–8.2.5

INVALIDATE_COLLISION **8.2.5**

KEEP_COPY **8.2.2**, 8.2.3

LAST_TRANSACTION_FLAG **6.2.1.4**, 8.2.2

LIVE_INSERTION **7.2.1**

LIVE_WITHDRAWAL_COMMAND **7.2.1**, 7.2.3.5

LOCK_CMD_0 **6.2.1.5**, 6.2.1.6, 6.2.2.2.1

LOCK_CMD_1 **6.2.1.5**, 6.2.1.6, 6.2.2.2.2

LOCK_CMD_2 **6.2.1.5**, 6.2.1.6, 6.2.2.2.3

LOCK_HOLD 6.2.1.2, **6.2.1.5**, 6.2.2.2.1, 6.2.3.23

LOCKED 6.2.1.1, **6.2.1.5**, 6.2.3.2

LOW_SPEED **6.2.1.1**, 6.2.2.3.1

LOW_SPEED_STATUS **6.2.1.1**

MAILBOX_INITIALIZATION **9.2.1**

MAILBOX_SELECTED 6.2.1.7.2, 6.2.2.4.5, **9.2.1**

MASK_SWAP_CAPABLE **7.2.4.1.1**

MASK_SWAP **6.2.1.5**, 6.2.3.3–6.2.3.8, 7.2.4.1.1

MASTER 5.2.3, **6.2.1.1**, 6.2.1.2–6.2.1.6, 6.2.1.7.1, 6.2.1.7.2, 6.2.1.8, 6.2.2.1.1, 6.2.2.1.4, 6.2.2.2.1–6.2.2.2.9, 6.2.2.5.1–6.2.2.5.3, 6.2.2.5.6, 6.2.3.1, 6.2.3.3, 6.2.3.5, 6.2.3.7, 6.2.3.9, 6.2.3.11, 6.2.3.13, 6.2.3.15, 6.2.3.17, 6.2.3.19, 6.2.3.22, 6.2.3.24, 7.2.4.1.1, 8.2.2–8.2.5, 9.2.1

MASTER_CAPABILITY_ERROR **6.2.1.7.1**, 6.2.1.8

MSTER_CAPABLE **7.2.4.1.1**

MASTER_ENABLE 4.2.2.1, 4.2.2.2, 5.2.3, **7.2.4.2.2**

MEMORY **8.2.1**

MEMORY_AGENT **8.2.1**, 8.2.2, 8.2.4, 8.2.5

MESSAGE_BUSY 6.2.1.7.2, **9.2.1**

MESSAGE_CAPABLE **7.2.4.1.1**

MESSAGE_CONFLICT 6.2.2.4.5, **9.2.1**

MODIFIED_RESPONSE **6.2.1.4**, 6.2.1.7.2, 6.2.2.5.1, 6.2.3.1, 8.2.2–8.2.5

MORE **6.2.1.4**, 6.2.2.2.2

MULTIPLE_PACKET_CAPABLE **7.2.4.1.1**

MULTIPLE_PACKET_MODE **6.2.1.1**, 6.2.1.4, 6.2.1.7.1, 6.2.1.7.2, 6.2.1.8, 6.2.3.1, 6.2.3.2, 8.2.2

MULTIPLE_PACKET_MODE_ENABLE 6.2.1.1, **7.2.4.2.1**

NO_ARBITRATION_OWNER **5.2.3**, 5.2.7.1.3, 5.2.8.5, 5.2.8.6

PACKET_CAPABLE 6.2.1.1, **7.2.4.1.1**

PACKET_ENABLED **6.2.1.1**

PACKET_LENGTH_2_CAPABLE **7.2.4.1.1**

PACKET_LENGTH_2_ENABLE 6.2.1.1, **7.2.4.2.1**

PACKET_LENGTH_4_CAPABLE **7.2.4.1.1**

PACKET_LENGTH_4_ENABLE 6.2.1.1, **7.2.4.2.1**

PACKET_LENGTH_8_CAPABLE **7.2.4.1.1**

PACKET_LENGTH_8_ENABLE 6.2.1.1, **7.2.4.2.1**

PACKET_LENGTH_16_CAPABLE **7.2.4.1.1**

PACKET_LENGTH_16_ENABLE 6.2.1.1, **7.2.4.2.1**

PACKET_LENGTH_32_CAPABLE **7.2.4.1.1**

PACKET_LENGTH_32_ENABLE 6.2.1.1, **7.2.4.2.1**

PACKET_LENGTH_64_CAPABLE **7.2.4.1.1**

PACKET_LENGTH_64_ENABLE 6.2.1.1, **7.2.4.2.1**

PACKET_NAK **6.2.1.7.2**, 6.2.2.4.2

PACKET_REQUEST **6.2.1.1**, 6.2.1.7.2, 6.2.2.2.1

PARITY_REPORT_ENABLE 5.2.6, 6.2.1.7.2, **7.2.4.2.2**

PARTIAL **6.2.1.4**, 6.2.1.7.2, 6.2.1.8, 6.2.2.5.16.2.1.7.2–6.2.2.5.3, 6.2.3.3–6.2.3.8

PARTICIPATING 6.2.1.1–6.2.1.6, **6.2.1.7.2**, 6.2.1.8, 6.2.2.1.5, 6.2.2.1.6, 6.2.2.5.1–6.2.2.5.3, 6.2.3.2, 8.2.2

PHASE_0 5.2.1–5.2.4, **5.2.5**, 5.2.6, 5.2.7.1.1, 5.2.7.2.1, 5.2.7.2.2

PHASE_1 5.2.1–5.2.4, **5.2.5**, 5.2.7.1.1, 5.2.7.1.3

PHASE_2 **5.2.5**, 5.2.6 5.2.7.1.2, 5.2.8.3, 5.2.9.3

PHASE_3 5.2.2–5.2.4, **5.2.5**, 5.2.6, 5.2.7.1.1, 5.2.7.1.2, 5.2.7.2.1
 PHASE_4 5.2.4, **5.2.5**, 5.2.6, 5.2.7.1.3, 5.2.8.5, 5.2.9.5
 PHASE_5 5.2.1–5.2.4 **5.2.5**, 5.2.7.1.2, 5.2.7.1.3, 6.2.1.5, 7.2.1
 POWER_UP 4.2.2.4, 5.2.7.1.3, 6.2.2.1.3, **7.2.1**, 7.2.3.1, 7.2.4.2.1–7.2.4.2.6
 POWERUP_TIME **7.2.1**, 7.2.3.2
 PREVENT_EU **8.2.2**
 QUEUED_TRANSACTION_FLAG **8.2.2**, 8.2.3
 READ_INVALID **6.2.1.4**, 6.2.1.8, 6.2.3.1, 8.2.2–8.2.5
 READ_LOCKED **6.2.1.4**, 6.2.1.5, 6.2.1.8, 6.2.3.1
 READ_MODIFIED **6.2.1.4**, 6.2.1.8, 6.2.3.1, 8.2.2–8.2.5
 READ_PARTIAL **6.2.1.4**, 6.2.1.8, 6.2.3.1
 READ_PARTIAL_LOCKED **6.2.1.4**, 6.2.1.5, 6.2.1.5, 6.2.3.1
 READ_RESPONSE 6.2.1.3, **6.2.1.4**, 6.2.2.5.1, 6.2.2.5.3, 6.2.3.1
 READ_SHARED **6.2.1.4**, 6.2.1.8, 6.2.3.1, 8.2.2–8.2.5
 READ_UNLOCKED **6.2.1.4**, 6.2.1.8, 6.2.2.2.2, 6.2.3.1
 READY_FOR_WITHDRAWAL 6.2.3.2, **7.2.1**, 7.2.3.5
 REQUEST_MAILBOX **9.2.1**
 REQUEST_MAILBOX_BUSY **9.2.1**
 REQUEST_MAILBOX_CONFLICT **9.2.1**
 REQUEST_MAILBOX_SELECTED **9.2.1**
 REQUESTER 6.2.1.7.2, 8.2.2, **8.2.4**
 REQUESTER_EXCLUSIVE 8.2.2, 8.2.3, **8.2.4**
 REQUESTER_EXCLUSIVE_STATUS **8.2.2**
 REQUESTER_SHARED 8.2.2, 8.2.3, **8.2.4**
 REQUESTER_SHARED_STATUS **8.2.2**
 REQUESTER_STATUS **8.2.2**
 REQUESTER_WAITING 8.2.2, **8.2.4**
 REQUESTER_WAITING_STATUS **8.2.2**
 REQUESTER_WRITE_INVALID 8.2.2, **8.2.4**
 REQUESTER_WRITE_INVALID_STATUS **8.2.2**
 RESET_COMPLETE **7.2.1**, 7.2.2
 RESET_EVENT **7.2.1**

RESOURCE_LOCK **6.2.1.5**, 6.2.3.2, 6.2.3.23

RESPONDER 8.2.2, **8.2.5**

RESPONDER_EXCLUSIVE 8.2.2, **8.2.5**

RESPONDER_EXCLUSIVE_STATUS **8.2.2**

RESPONDER_INVALIDATE 8.2.2, **8.2.5**

RESPONDER_INVALIDATE_STATUS **8.2.2**, 8.2.5

RESPONDER_SHARED 8.2.2, **8.2.5**

RESPONDER_SHARED_STATUS **8.2.2**

RESPONDER_STATUS **8.2.2**

RESPONDER_WRITE_INVALID 8.2.2, **8.2.5**

RESPONDER_WRITE_INVALID_STATUS **8.2.2**

RESPONSE_MAILBOX **9.2.1**

RESPONSE_MAILBOX_BUSY **9.2.1**

RESPONSE_MAILBOX_CONFLICT **9.2.1**

RESPONSE_MAILBOX_SELECTED **9.2.1**

ROUND_ROBIN **5.2.3**, 5.2.4, 5.2.8.2, 5.2.8.6

RQ **5.2.1**, 5.2.9.2

SECOND_PASS 5.2.1–5.2.3, **5.2.4**, 5.2.8.1, 5.2.8.2, 5.2.8.4, 5.2.8.6, 5.2.9.2, 5.2.9.6

SECOND_PASS_REQUIRED 5.2.1–5.2.3, **5.2.4**, 5.2.8.1, 5.2.8.4, 5.2.8.6, 5.2.9.1, 5.2.9.4, 5.2.9.6

SELECTED 6.2.1.5, **6.2.1.7.2**, 6.2.2.4.3, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, 6.2.3.14, 6.2.3.16, 6.2.3.18

SELECTED_STATUS **6.2.1.7.1**

SETTLED **5.2.5**, 5.2.8.1, 5.2.8.3, 5.2.9.1, 5.2.9.3

SHARED_RESPONSE **6.2.1.4**, 6.2.1.7.2, 6.2.2.5.1, 6.2.3.1, 8.2.2–8.2.5

SHARED_UNMODIFIED 8.2.2, **8.2.3**

SHARED_UNMODIFIED_STATUS **8.2.2**, 8.2.3

SLAVE_WRITE **6.2.1.4**, 6.2.1.8, 6.2.2.1.5, 6.2.2.1.6, 6.2.2.5.1–6.2.2.5.3, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, 6.2.3.14, 6.2.3.16, 6.2.3.18

SNARF_DATA 6.2.2.4.4, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, **8.2.2**, 8.2.3

SPLIT **6.2.1.2**, 6.2.2.3.3, 8.2.3, 8.2.5

SPLIT_ACCEPTOR **7.2.4.1.1**, 7.2.4.2.1, 8.2.1

SPLIT_CACHED 6.2.1.2, **8.2.2**

SPLIT_ENABLE 6.2.1.2, **7.2.4.2.1**, 8.2.1

SPLIT_INITIATOR 6.2.1.2, **7.2.4.1.1**

SPLIT_STATUS **6.2.1.2**, 6.2.1.6, 6.2.1.8, 6.2.2.2.2, 6.2.2.5.1, 6.2.3.3, 6.2.3.22, 6.2.3.23, 8.2.3–8.2.5

SYS_RESET 4.2.2.4, **7.2.1**, 7.2.2, 7.2.3.1–7.2.3.4, 7.2.4.2.1–7.2.4.2.6, 8.2.3–8.2.5

TAG_CAPABLE **7.2.4.1.1**

TAG_ENABLE 6.2.2.5.5, **7.2.4.2.2**

TRANS_CMD_0 **6.2.1.4**, 6.2.2.2.1

TRANS_CMD_1 **6.2.1.4**, 6.2.2.2.2

TRANS_CMD_2 **6.2.1.4**, 6.2.2.2.3

TRANS_CMD_3 **6.2.1.4**, 6.2.2.2.4

TRANSACTION_BEGIN 5.2.4, 5.2.8.5, **6.2.1.8**, 6.2.2.1.1, 6.2.3.1

TRANSACTION_COMPLETE 5.2.7.1.3, 5.2.8.5, 6.2.1.1–6.2.1.6, 6.2.1.7.1, 6.2.1.7.2, **6.2.1.8**, 6.2.2.1.1, 6.2.2.1.3, 6.2.3.22, 6.2.3.23, 8.2.2, 9.2.1

TRANSACTION_ERROR **6.2.1.7.2**, 6.2.1.8, 6.2.2.4.1, 8.2.2

TRANSACTION_ERROR_STATUS **6.2.1.7.2**

TRANSACTION_FLAG_STATUS **6.2.1.7.1**, 6.2.2.2.2, 8.2.3

TRANSACTION_TIMEOUT **6.2.1.8**, 6.2.3.1, 6.2.3.3, 6.2.3.5, 6.2.3.7, 6.2.3.9, 6.2.3.11, 6.2.3.13, 6.2.3.15, 6.2.3.17, 6.2.3.22, 6.2.3.24, 7.2.4.2.5

TRANSFER_1 **6.2.1.6**, 6.2.1.8

TRANSFER_2 **6.2.1.6**, 6.2.1.8, 7.2.4.1.1

TRANSFER_4 **6.2.1.6**, 6.2.1.8, 7.2.4.1.1

TRANSFER_8 **6.2.1.6**, 6.2.1.8, 7.2.4.1.1

TRANSFER_16 **6.2.1.6**, 6.2.1.8, 7.2.4.1.1

TRANSFER_32 **6.2.1.6**, 6.2.1.8, 7.2.4.1.1

TRANSFER_64 **6.2.1.6**, 6.2.1.8, 7.2.4.1.1

TRANSFER_COUNT_ZERO 6.2.1.1, **6.2.1.8**, 6.2.2.1.1, 6.2.2.1.4, 6.2.3.1, 6.2.3.3, 6.2.3.5, 6.2.3.7, 6.2.3.9, 6.2.3.11, 6.2.3.15, 6.2.3.17

TRANSFER_UNRESTRICTED **6.2.1.6**

TWO_PASS **5.2.3**, 5.2.4, 5.2.8.2

UNIQUENESS_GREATER **5.2.3**, 5.2.8.1, 5.2.8.4

WAIT **6.2.1.7.2**, 6.2.2.4.8

WAIT_CACHED 6.2.1.7.2, **8.2.2**

WAIT_STATUS 5.2.3, **6.2.1.7.1**, 6.2.1.8, 8.2.3, 8.2.4

WINNER 5.2.1–5.2.4, **5.2.5**, 5.2.7.1.2, 5.2.7.1.3, 5.2.8.1, 5.2.8.3, 5.2.8.5, 5.2.8.6, 5.2.9.1, 5.2.9.3, 5.2.9.6

WRITE_CMD **6.2.1.4**, 6.2.1.6, 6.2.1.7.2, 6.2.2.2.5, 6.2.2.5.1, 6.2.2.5.2, 6.2.2.5.3, 6.2.3.3, 6.2.3.5, 6.2.3.7, 6.2.3.9, 6.2.3.11, 6.2.3.13, 6.2.3.15, 6.2.3.17, 6.2.3.22, 6.2.3.23

WRITE_INVALID **6.2.1.4**, 6.2.2.5.1, 6.2.3.1, 8.2.2–8.2.5

WRITE_LOCKED **6.2.1.4**, 6.2.1.5, 6.2.2.5.1, 6.2.3.1

WRITE_NO_ACKNOWLEDGE **6.2.1.4**, 6.2.2.5.1, 6.2.3.1, 9.2.1, 9.2.2.1–9.2.2.14

WRITE_PARTIAL **6.2.1.4**, 6.2.2.5.1, 6.2.3.1

WRITE_PARTIAL_LOCKED **6.2.1.4**, 6.2.1.5, 6.2.2.5.1, 6.2.3.1

WRITE_RESPONSE 6.2.1.3, **6.2.1.4**, 6.2.1.8, 6.2.2.5.1, 6.2.2.5.3

WRITE_STATUS **6.2.1.4**, 6.2.1.7.2, 6.2.3.4, 6.2.3.6, 6.2.3.8, 6.2.3.10, 6.2.3.12, 6.2.3.14, 6.2.3.16, 6.2.3.18

WRITE_UNLOCKED **6.2.1.4**, 6.2.2.2.2, 6.2.2.5.1, 6.2.3.1, 9.2.1, 9.2.2.1–9.2.2.14

2.8 Implementation mnemonics

Data sheets for products based on this International Standard shall use the following mnemonics to indicate capabilities:

A32	32-bit addressing implemented.
A64	64-bit addressing implemented.
CACHE	The cache coherence protocol is implemented.
CARB	A central arbiter interface is implemented.
DARB	A distributed arbiter interface is implemented.
D32	Data path width of 32 bits.
D64	Data path width of 64 bits.
D128	Data path width of 128 bits.
D256	Data path width of 256 bits.
PKTn	Able to use packet transfer mode at a clock rate of n MHz. If more than one frequency is supported, multiple PKTn mnemonics shall be listed.
MSG	Message passing protocol is implemented.
MSTR	Capable of becoming bus master.
MSLOCK	Able to service mask and swap locked commands.
CSLOCK	Able to service compare and swap locked commands.
FABLOCK	Able to service fetch and add big endian locked commands.
FALLOCK	Able to service fetch and add little endian locked commands.
SPLT	Able to service split transactions.
SPLTING	Able to service and initiate split transactions.
TAGn	Tag lines are implemented. The number of tag lines implemented is n.

A module able to become bus master, with a 64-bit data path width, 32-bit address capability, cache coherence mechanism, and able to service split transactions, would be A32/CACHE/D64/MSTR/SPLT.

3. Bus signaling environment

3.1 Description

This clause describes and specifies the signaling environment with which Futurebus+ implementations must comply. These specifications are *independent* of the signaling technology used: Futurebus+ may be implemented with any technology (TTL, Backplane Transceiver Logic, ECL, CMOS, or GaAs), provided the condition of incident wave switching and the specified constraints on skew are met. In order to ensure compatibility between products within a set described by a profile, particular logic families may be associated with specific profiles in IEEE Std 896.2-1991.

3.1.1 Incident wave switching

Incident wave switching is a requirement for all Futurebus+ systems. The requirement is as follows: The parameters of current, voltage, and characteristic impedance, under all conditions of module loading, shall always enable the first voltage step launched by a driver down the transmission line to exceed the threshold required to trigger correct reception by the receiver, on both positive and negative transitions, and shall guarantee that subsequent reflections do not re-cross the receiver thresholds. Typically, this is related to the ratio of the current capable of being driven by a device, to the capacitance it presents to a bus line when the driver is disabled.

3.1.2 Skew

Skew is the variance in the propagation time of two or more signals that may travel over different paths, but share a common source and destination. Two signal transitions can never be guaranteed to be exactly simultaneous. Therefore, the bus protocol must always allow for a skew between them.

A critical skew in a Futurebus+ system is measured between information signals and the synchronization signal that marks their validity on the bus. The sender module's designer must ensure that transmitted information signals arrive at the connection to the bus before the validating synchronization signal occurs. This skew requirement must take into account any skews introduced by the bus transmitters themselves, and the signaling medium.

In addition, the designer of the signaling medium must guarantee that the skew introduced by capacitive loading does not delay the information signals more than the synchronization signals. A module that is logically completely passive is still electrically active with regard to bus skew. The length of the PCB trace stubs, through-plated holes, and the transceivers themselves contribute to the loading.

The designer of receiving modules must also delay the synchronization signal to account for any skew in the bus receivers as the information is received. This will ensure that the cause-and-effect relationship between data is preserved as the signals are provided to the first synchronization barrier within the receiving module.

3.2 Specification

3.2.1 Skew

The bus shall exhibit a skew such that the propagation delays of the signal lines to the left of the relational operator in relationships 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, and 3.8 are greater than or equal to the propagation delays of the signal lines to the right of the relational operator in these relationships:

All modules shall restrict the capacitive loading skew such that the relative loading on the signal lines exhibited by the module meets the relationships 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, and 3.8 above.

$$(AS^*, DS^*) \geq (AD[]^*, D[]^*, BP[]^*, TG[]^*, TP^*, CM[]^*, CP^*) \quad (3.1)$$

$$AK^* \geq ST[]^* \quad (3.2)$$

$$AI^* \geq CA[]^*, ST[]^* \quad (3.3)$$

$$(DK^*, DI^*) \geq (AD[]^*, D[]^*, BP[]^*, TG[]^*, TP^*, ST[]^*) \quad (3.4)$$

$$(AP^*, AQ^*, AR^*) \geq (AB[]^*, AC[]^*, ABP^*) \quad (3.5)$$

$$DS^* \geq DK^* \quad (3.6)$$

$$AS^* \geq (AK^* AI^*) \quad (3.7)$$

$$GR^* \geq PE^* \quad (3.8)$$

3.2.2 Glitch filters

Glitch filters shall be provided if necessary to filter out a fundamental transmission line effect called the wire-OR glitch. It is anticipated that they will be needed for AK^* , AI^* , DK^* , DI^* , AP^* , AQ^* , AR^* , and RE^* . This specification is written assuming they are needed. The document specifying the electrical environment will specify whether they are needed for a particular signaling environment. For example, IEEE Std 896.2-1991 mandates the use of glitch filters in systems implemented using the signaling environment of Bus Transceiver Logic.

4. Centralized arbitration

4.1 Description

When a module needs to send data to or obtain data from another module, it must first gain tenure of the parallel bus. Since two or more modules may seek to gain tenure of the bus at the same time, an arbitration process is used to restrict tenure of the bus to one module at a time.

The central arbiter scheme involves a starred request/grant signal set from each requesting module to a central arbiter facility, which resolves simultaneous requests and serializes grants to the requesting modules.

This clause describes a simple centralized arbitration protocol that modules may use to gain tenure of the bus. It allows high-speed operation and low request-to-grant latency. A distributed arbitration protocol is specified in clause 5.

4.1.1 Bus lines used for centralized arbitration

4.1.1.1 RQ0* request 0

A module asserts $rq0^*$ to request bus mastership at one of its two priorities. Modules maintain $rq0^*$ asserted until the last transaction of their bus tenure.

Modules may assert $rq0^*$ even if they are asserting $rq1^*$. This possibly allows an increase of their request priority. A module maintains both $rq0^*$ and $rq1^*$ asserted until it becomes bus master and is in its last transaction on the parallel bus, or until it receives GR^* and the internal request has been cancelled. Modules release $rq0^*$ and $rq1^*$ simultaneously.

4.1.1.2 RQ1* request 1

A module asserts $rq1^*$ to request bus mastership at one of its two priorities. Modules maintain $rq1^*$ asserted until the last transaction of their bus tenure.

Modules may assert $\mathbf{rq1}^*$ even if they are asserting $\mathbf{rq0}^*$. This possibly allows an increase of their request priority. A module maintains both $\mathbf{rq0}^*$ and $\mathbf{rq1}^*$ asserted until it becomes bus master and is in its last transaction on the parallel bus, or until it receives $\mathbf{GR}^*$ and the internal request has been cancelled. Modules release $\mathbf{rq0}^*$ and $\mathbf{rq1}^*$ simultaneously. Modules should not perform low-priority transactions during a tenure that resulted from a high-priority request. Modules should release the bus and request it again at the lower priority to avoid priority inversion.

4.1.1.3 $\mathbf{GR}^*$ grant

The central arbiter asserts $\mathbf{gr}^*$ in response to the assertion of $\mathbf{RQ0}^*$ or $\mathbf{RQ1}^*$ to indicate to a module that it is the master elect. The module may then begin parallel bus transactions when it detects that $\mathbf{ET}^*$ is released. The central arbiter maintains the assertion of $\mathbf{gr}^*$ until both $\mathbf{RQ0}^*$ and $\mathbf{RQ1}^*$ from the master have been released.

4.1.1.4 $\mathbf{PE}^*$ preempt

The central arbiter asserts $\mathbf{pe}^*$ to indicate to the master that a preemptive condition exists. The central arbiter releases $\mathbf{pe}^*$ when it releases $\mathbf{gr}^*$. The central arbiter resolves any metastability problems on the central arbiter occurring if $\mathbf{PE}^*$ is asserted near the time that the master releases its request.

Preemptive conditions would be defined by the central arbiter design. The central arbiter might assert $\mathbf{PE}^*$ if other modules are requesting at equal or higher priority. A different central arbiter design could assert $\mathbf{PE}^*$ if any other module was requesting the bus.

The central arbiter asserts $\mathbf{pe}^*$ during power-up and system reset to indicate to monarch-capable modules that there is a central arbiter in the system. Modules that are not monarch-capable are commanded to use the central arbiter through the `CENTRAL_ARBITER` bit in the Logical Common Control register (7.1.2.2.1).

4.1.2 Centralized arbitration operation

The centralized arbitration competition shown in figure 4 proceeds as follows:

- 1) Modules A and B each assert their arbitration request. For this example, Module A's request is a higher priority than Module B's request.
- 2) The central arbiter asserts $\mathbf{gr}^*$ to Module A.
- 3) The current master, Module A, asserts $\mathbf{et}^*$ after beginning a transaction.
- 4) During its last transaction, Module A finishes its need for bus tenure and releases its request.
- 5) Upon detecting that Module A has released its request, the central arbiter releases the grant to Module A and asserts the grant to Module B. Module B, at that point, becomes the master elect.
- 6) The master, Module A, after releasing the parallel bus signals, releases $\mathbf{et}^*$ to indicate to the master elect that it is the new master.
- 7) After beginning its transaction the current master, Module B, asserts $\mathbf{et}^*$.
- 8) During its last transaction, Module B finishes its need for bus tenure and releases its request.
- 9) Upon detecting that Module B has released its request, the central arbiter releases the grant to Module B. Since no requests are pending, the central arbiter issues no grant.
- 10) When the current master, Module B, has released the parallel bus signals, it releases $\mathbf{et}^*$.

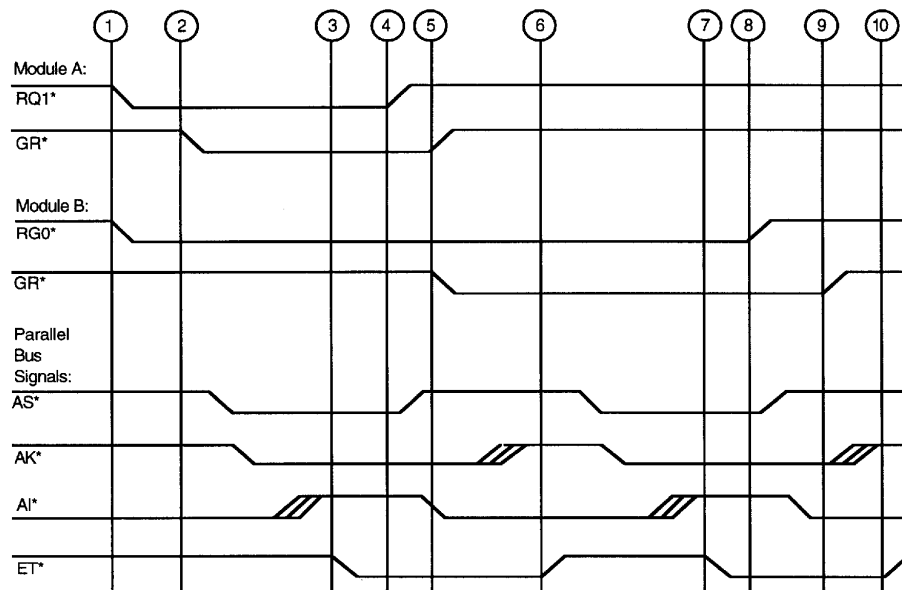


Figure 4— Two-competitor centralized arbitration

4.1.3 Central arbiter description

The central arbiter has few mandatory requirements. Since many different types of arbiters are possible that will allow interoperability, most requirements are the responsibility and privilege of the system integrator to specify.

Any number of bus allocation schemes may be implemented. Some possibilities may be:

- A simple, single-priority, first-come-first-served arbiter.
- A two-priority arbiter with fairness allocation within each priority level.
- A more complex scheme where each request is assigned one of 256 priorities. Priorities would be changed by modules sending messages to the central arbiter using the arbitrated message facilities.

4.2 Specification

4.2.1 Bus arbitration attributes

CENTRALIZED_MASTER. A module shall set CENTRALIZED_MASTER when $\sim\text{INIT} \& (\text{rq0}^* \mid \text{rq1}^*) \& \text{GR}^*$ & $\sim\text{ET}^*$ and maintain it set until $\text{INIT} \mid \sim\text{rq0}^* \& \sim\text{rq1}^* \& \sim\text{et}^*$.

4.2.2 Bus arbitration signals

4.2.2.1 RQ0*

A module may assert rq0^* if $\sim\text{INIT} \& \text{MASTER_ENABLE} \& \text{CENTRAL_ARBITER} \& \sim\text{GR}^*$ to request mastership of the bus and shall maintain it asserted until $\text{INIT} \mid \text{GR}^* \& \text{et}^*$ during the master's last transaction of this tenure $\mid \text{GR}^*$ and the module chooses not to perform any transactions during this tenure.

4.2.2.2 RQ1*

A module may assert rq1^* if $\sim\text{INIT} \& \text{MASTER_ENABLE} \& \text{CENTRAL_ARBITER} \& \sim\text{GR}^*$ to request mastership of the bus and shall maintain it asserted until $\text{INIT} \mid \text{GR}^* \& \text{et}^*$ during the master's last transaction of this tenure $\mid \text{GR}^*$ and the module chooses not to perform any transactions during this tenure.

4.2.2.3 GR*

The central arbiter asserts gr^* to indicate to a module that it is the master elect. The central arbiter shall only assert gr^* to one module at a time. The central arbiter shall release gr^* to the **CENTRALIZED_MASTER** when it detects $\sim\text{RQ1}^*$ & $\sim\text{RQ0}^*$.

4.2.2.4 PE*

The central arbiter shall assert pe^* if it detects a preemptive condition, and shall maintain it asserted until it releases gr^* . The central arbiter shall release pe^* prior to asserting gr^* to another module. If the central arbiter needs to assert pe^* for consecutive tenures, it may choose to ignore the previous requirement and leave pe^* asserted from the assertion of one gr^* to the next.

The central arbiter shall assert pe^* if $\text{POWER_UP} \mid \text{SYS_RESET}$ and shall maintain it asserted for a minimum of 1 μs after it detects **REf** at logic zero.

NOTE — The detection of PE^* asserted by the central arbiter is subject to metastability. The pulse width of PE^* may be shorter than that required by a module. Modules must allow for these conditions and operate correctly.

5. Distributed arbitration and arbitrated messages

5.1 Description

When a module needs to send data to or obtain data from another module, it must first gain tenure of the parallel bus. Since two or more modules may seek to gain tenure of the bus at the same time, an arbitration process is used to restrict tenure of the bus to one module at a time. Futurebus+ provides the user with the choice of two methods to select the next bus master.

The central arbitration protocol is described in clause 4. The distributed arbitration protocol is described in this clause. In either case, the distributed protocol is used to broadcast arbitrated messages to one or more modules, or to the central arbiter.

5.1.1 Arbitrated messages-central arbiter

When the central arbiter is used, the arbitration process is used solely for the transmission of arbitrated messages. Two types of messages are provided, general arbitrated messages and central arbiter messages. General messages are usually directed at other modules on the bus, while central messages are used to control module request priorities in the central arbiter.

Some examples of uses for this messaging facility include sending power fail and other asynchronous broadcast interrupts, asynchronous targeted interrupts, software synchronization barriers, retry events, and bidding to supply services.

5.1.1.1 General arbitrated messages

Modules use a message number to resolve an arbitration competition. When two or more modules arbitrate to send a message, the winner is the one whose message number is greatest. Losing modules initiate a new arbitration competition to try again. The mechanism used is called parallel contention arbitration, a process whereby modules assert their message numbers on the arbitration bus and release signals according to an algorithm that, after a period of time, will ensure that only the winner's number remains on the arbitration bus.

General arbitrated messages are only 7 bits, so a unique number can be selected in a single competition. Each cycle consists of a competition using an 8-bit competition number, **cn[7..0]**. A module wishing to deliver a general arbitrated message uses the competition number shown in table 2. The most significant bit is a 1, which ensures that general arbitrated messages have higher priority than central messages.

Table 2— General arbitrated message fields

General arbitrated messages (central arbiter)								
Bit	cn7	cn6	cn5	cn4	cn3	cn2	cn1	cn0
Pass 1	1	AM6	AM5	AM4	AM3	AM2	AM1	AM0

5.1.1.2 Central arbitrated messages

Central arbitrated messages are 14 bits, and are intended primarily to send priority information to the central arbiter. Central arbiter messages require two competition cycles in order to allow up to 256 priority levels to be used in a system. New message requests cannot be initiated between the first and second pass of a two-pass message transmission. Although there can be more than one winner after the first pass, the second pass will choose a unique message.

A central arbiter message consists of three fields, as shown in table 3. The priority field, **PR[7..0]**, selects the priority level of the module's request line. This priority level is applied to the request line corresponding to the request select field, **RQ**. If **RQ** is zero, the **RQ0*** line is selected. If **RQ** is one, the **RQ1*** line is selected. The geographical address field, **GA[4..0]**, identifies the module requesting the updated request level. In general, a module will control its own priority level, but it is possible for one module to take responsibility for another's priorities. The most significant bit of the competition number determines whether the message is in the first or last pass, and ensures that central arbitrated messages have lower priority than general ones.

Table 3— Central arbitrated message fields

Central arbitrated messages (central arbiter)								
Bit	cn7	cn6	cn5	cn4	cn3	cn2	cn1	cn0
Pass 1	0	PR7	PR6	PR5	PR4	PR3	PR2	PR1
Pass 2	1	PRO	RQ	GA4	GA3	GA2	GA1	GA0

5.1.2 Arbitration and messages—distributed arbiter

Systems with no central arbiter use a distributed arbitration protocol. In this case, the arbitration process is used for both bus requests and arbitrated messages. The distributed arbitration requests allow modules to obtain bus tenure. The distributed arbitration messages are similar to the general arbitrated messages above, directed at other modules on the bus.

5.1.2.1 Distributed arbitration messages

Distributed arbitration messages are 7 bits consisting of bits **AM[6..0]**. However, they require a two-pass competition because a first pass of all 1's is needed to distinguish them from distributed arbitration requests, and to ensure that those requests do not interfere with the transmission of emergency information. A module wishing to deliver a

distributed arbitration message uses the competition number shown in table 4. During the second pass, the most significant bit is set to 1 for consistency with the general arbitrated message format.

Table 4— Distributed arbitration message fields

Distributed arbitration messages (distributed arbiter)								
Bit	cn7	cn6	cn5	cn4	cn3	cn2	cn1	cn0
Pass 1	1	1	1	1	1	1	1	1
Pass 2	1	AM6	AM5	AM4	AM3	AM2	AM1	AM0

5.1.2.2 Distributed arbitration requests

The value of the distributed arbitration request numbers used by modules in a competition determine the number of competition passes. Some numbers require two passes of the control acquisition cycle, while others require only a single pass. Each pass of the control acquisition cycle consists of an arbitration competition using the 8-bit competition number, **cn[7..0]**. More than one module may have the highest number after the first pass, but there is only a single winner at the end of the second pass. This is guaranteed because the competition numbers used in the second pass are unique. Arbitration numbers requiring only a single-control acquisition cycle may be freely and dynamically mixed with those requiring two-control acquisition cycles.

An arbitration number consists of three fields, as shown in table 5: the priority field, the round-robin field, and the geographical address field. A module's arbitration number maps onto one or two competition numbers. This mapping is purely a performance enhancement and does not imply that two separate classes of arbitration numbers exist. The bit **cn7** is used to indicate whether the module requires another arbitration pass, so that a single-pass arbitration number defeats a two-pass arbitration number if both occur in the same control acquisition cycle.

Table 5— Distributed arbitration request fields

Distributed arbitration requests (distributed arbiter)								
Bit	cn7	cn6	cn5	cn4	cn3	cn2	cn1	cn0
Single pass								
Pass 1	1	PRO	RR	GA4	GA3	GA2	GA1	GA0
Two pass								
Pass 1	0	PR7	PR6	PR5	PR4	PR3	PR2	PR1
Pass 2	1	PRO	RR	GA4	GA3	GA2	GA1	GA0

5.1.2.2.1 Priority field

The 8-bit priority field, **PR[7..0]**, selects one of 256 priority levels for the request. A module's priority class can be set by the system or the module. The highest two priority levels, where all but **PRO** are 1's, use the single pass competition cycle. This is possible because when **PR[7..1]** are all 1's the requester would always win a first pass, so it can be skipped for this case. Therefore, the priorities 254 and 255 have almost twice the request-to-grant performance of the other priorities, so they should be used in systems that only use one or two priority levels.

5.1.2.2.2 Round-robin field

The single-bit round-robin field, **RR**, in conjunction with the geographical address field, determines the position of a module in a round-robin request queue. The round-robin protocol ensures fair and equitable allocation of bus tenure to competing modules by allowing all modules a turn at bus tenure, in a sequence defined by the magnitude of the unique field assigned to that module.

Modules using the round-robin protocol monitor arbitration competitions. The value of a module's round-robin bit is adjusted each time a transfer of tenure to another module in its priority class is observed, whether or not it competes

in that arbitration competition. The round-robin bit is set each time tenure of the bus is transferred to a module in the same priority class, but with a larger unique field value. Whenever tenure of the bus is transferred to a module in its priority class with a lesser unique field value, the round-robin bit is cleared.

Once a module has acquired tenure of the bus and cleared its round-robin bit, it is defeated in any subsequent arbitration competition by all modules competing with any geographical address field value including those that are lower. When all outstanding requests from modules with a lower value of geographical address field have been satisfied, it has a chance to regain bus tenure.

If a module using the round-robin protocol relinquishes its bus tenure to a module of higher priority class before its own transactions are complete, it may retain the value “1” for its round-robin bit, ensuring that it regains tenure of the bus as soon as it becomes available to a module of its priority class.

5.1.2.2.3 Geographical address field

The geographical address field is a five-bit field, **GA[4..0]**, which guarantees the uniqueness of the arbitration number. If a module is competing for the bus, this number must correspond to the geographical address of the module. The value “11111” is not allowed, due to a conflict with the CSR broadcast addressing scheme.

5.1.2.3 Depositing the master elect

In distributed arbitration, the module with the highest arbitration number at the end of the competition must wait until the master has finished performing bus transactions before it can become master. While waiting in this state, a module is referred to as the *master elect*.

There may be times when a module has urgent need of the bus after a master elect has been chosen, but before the master elect has become master. If that module has a higher arbitration number than the master elect, it may initiate a new competition to establish a new master elect. This process is referred to as *deposing the master elect*.

Deposing the master elect allows a high priority module to acquire tenure of the bus with minimum latency (although with some sacrifice to the overall performance of the system, since it forces a new arbitration competition).

5.1.2.4 Transaction preemption

Where a priority-based scheme is in operation, a module that is the initiator of a long block transaction or a series of related transactions that could take place within a single tenure of the bus may, in the presence of a master elect from a higher priority class, choose to voluntarily suspend its current transaction. This allows a scheme of preemptive bus scheduling to be implemented. The master obtains the information it needs on the new master elect by monitoring the identity and priority of arbitration winners while it simultaneously conducts its long transaction or series of transactions on the parallel bus.

If the master observes that a module of greater priority has become master elect, it should relinquish tenure of the bus to the higher priority module as soon as is practical.

5.1.2.5 Parking

If there has been no intervening bus request, a bus master retains control of the data bus even after it has finished its transactions. This state is known as parking. If the master has a new request while it still owns the data bus, it can begin a new data bus transaction immediately, without initiating an arbitration cycle. This has the potential to greatly reduce average bus latency in some types of systems.

5.1.3 Bus lines

5.1.3.1 AP^* , AQ^* , AR^* control acquisition synchronization

These lines keep modules synchronized throughout the control acquisition process. The lines operate in a six-state sequence referred to as phase 0 through phase 5.

5.1.3.2 $AC[1..0]^*$ arbitration condition

During phase 3 through phase 5, modules assert $AC0^*$ to signal an arbitration error.

In distributed arbitration, during phase 3 through phase 5, modules assert $AC1^*$ to inhibit the transfer of tenure at the completion of a control acquisition cycle. In a central arbiter system, modules assert $AC1^*$ to inhibit the transmission of a partial central arbitrated message which occurs during the first pass of the two-pass competition. $AC1^*$ is always asserted along with $AC0^*$ if an error occurs. If $AC1^*$ is asserted alone during phase 3 through phase 5, transfer of tenure is inhibited for reasons other than the detection of errors.

5.1.3.3 $AB[7..0]^*$ arbitration bus

Competing modules apply their competition numbers, $cn[]$, to $ab[]^*$ during phase 1. However, if any signal cnx is a logic zero, and the signal on the corresponding bus line ABx^* is a logic one, then for as long as this condition persists, all bits, $ab[x-1..0]^*$, of lower significance are released. This continues through phase 3.

5.1.3.4 ABP^* arbitration parity

If the number of ones in a competitor's competition number is even, the competitor must assert abp^* during phase 1. However, if any of the bits, $ab[]^*$, are released as described in 5.1.3.3, abp^* shall also be released. This continues through phase 3.

5.1.4 Arbitration competition logic

As shown in the sample circuits in figures 5a) and 5b), if a module's competition number has a bit position of 0 and another module has a 1 in that position, the module with the 0 releases all $ab[]^*$ lines of lower significance. Eventually the $AB[]^*$ lines carry the number of the module with the highest arbitration number.

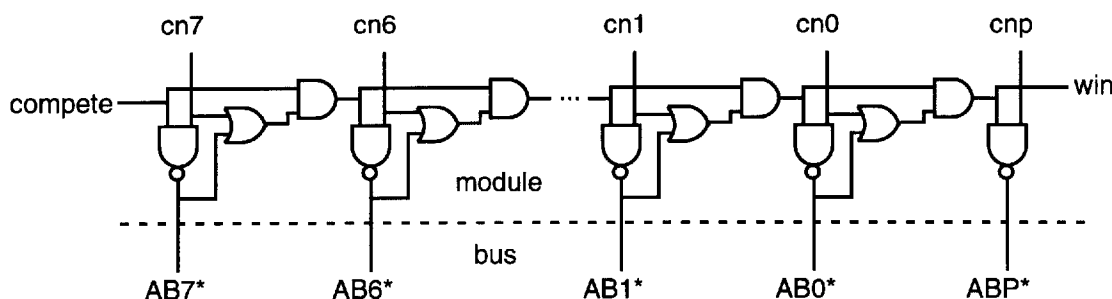


Figure 5a— Serial implementation of parallel contention logic`

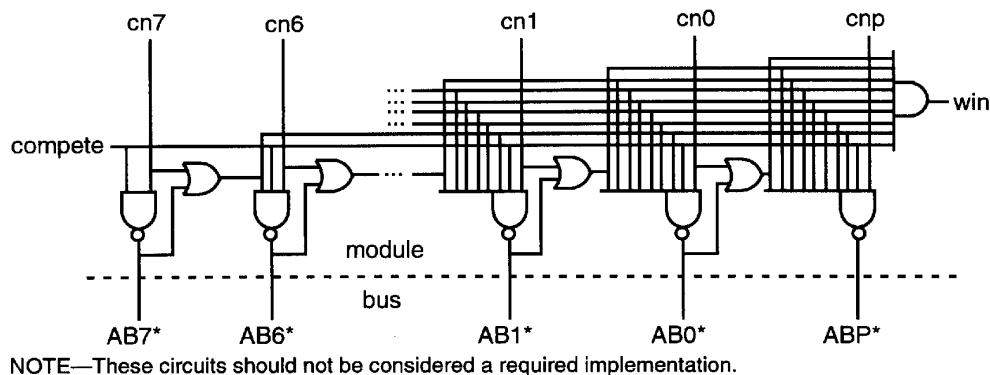


Figure 5b— Parallel implementation of parallel contention logic

In some cases, modules release **ab[]*** lines only to assert them at a later time. Figure 6 illustrates an arbitration competition where three modules simultaneously begin competition using the numbers “00001010,” “00001001,” and “00000100,” respectively. **AB[7..4]*** are not shown on the figure since they remain released throughout the sequence.

The arbitration competition in figure 6 proceeds as follows:

- 1) The three modules assert their competition numbers on the arbitration bus.
- 2) After a delay due to bus propagation, the bus lines become the wire-OR of the individual module lines causing **AB[3..0]*** to assert.
- 3) Module one, upon detecting **AB2*** asserted, releases **ab1***. Module two, upon detecting **AB2*** and **AB1*** asserted, releases **ab0***. Module three, upon detecting **AB3*** asserted, releases **ab2***.
- 4) The bus reflects the release of the bus lines.
- 5) Module 1, upon detecting **AB2*** released, asserts **ab1***. Module 2, upon detecting **AB2*** and **AB1*** released, asserts **ab0***.
- 6) The bus reflects the assertion of the bus lines.
- 7) Module 2 detects **AB1*** asserted and releases **AB0***.
- 8) The bus reflects the release of the bus line.

As shown in figure 6, the winning module (Module 1) releases **ab1*** and then later reasserts it. Module 1 also must wait for Module 2 to release **ab0*** before it knows it is the winner. The time required for the winning arbitration number to settle on the bus is referred to as the arbitration competition settling time.

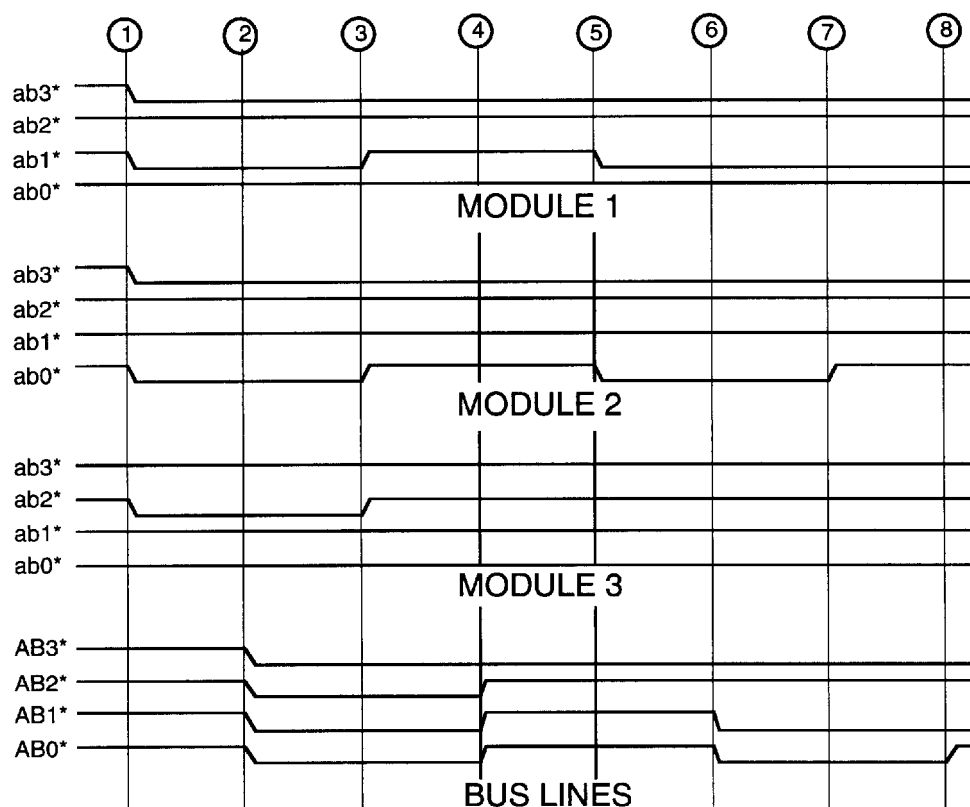


Figure 6 —Arbitration competition example

5.1.5 Arbitration competition settling time

The arbitration settling time is a function of the length of the backplane, the module's competition number, and the worst-case transceiver and logic delays of both the internal arbitration logic of the module and that of its competitors. The theoretical calculation of the worst-case arbitration settling time can be extremely complex, and is beyond the scope of this International Standard. The worst possible settling time for any competition number, however, can be given by the following equation:

$$t_a = 10 * t_{pd'} + 5 * t_{ext} + 4 * t_{int} + t_{win}$$

The worst-case delay can be considerably lower for most competition numbers, so if the performance of the distributed arbiter is of concern, it is worth trying to optimize this equation for each module's requests and messages.

The symbol $t_{pd'}$ is used to represent the maximum end-to-end signal propagation delay along the transmission lines which constitute the logical bus. The value for $t_{pd'}$ is in the Bus Propagation Delay register described in 7.1, and is a function of the maximum length of the transmission line in the application system.

The symbol t_{int} is used to represent the delay between a change on any arbitration line, ABx^* , on the input to a module and the corresponding change on that module's adjacent arbitration output, $ab[x-1]^*$, or abp^* , as its competition number is applied to or withdrawn from those bits.

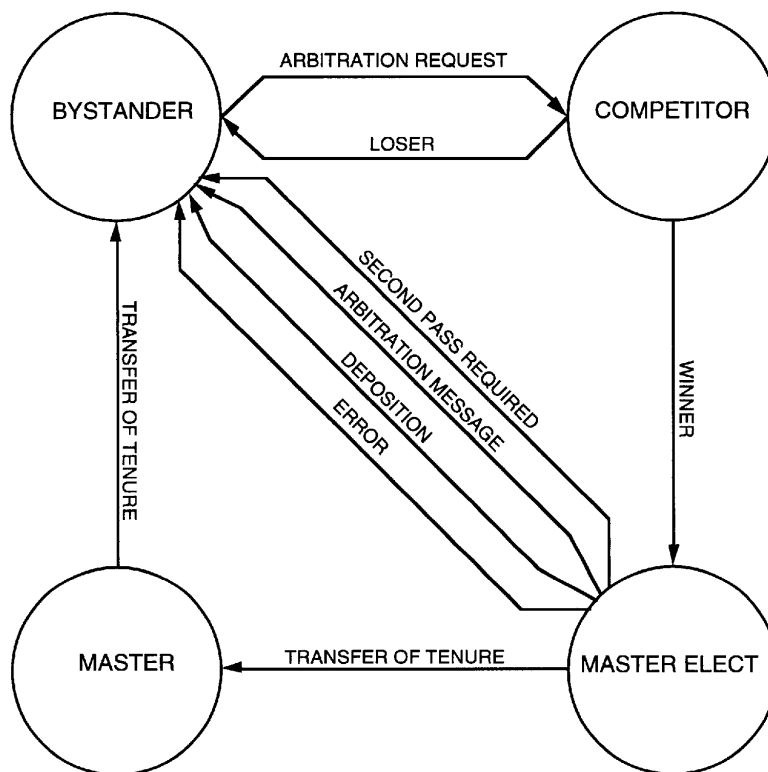
The symbol t_{ext} is used to represent the delay between a change on any arbitration line, $\mathbf{ABx}^*$, on the input of any other competing module and the corresponding changes on that module's arbitration outputs, $\mathbf{ab[x-1]}^*$, or $\mathbf{abp}^*$, as its competition number is applied to or withdrawn from those bits.

The symbol t_{win} is used to represent the maximum delay between the time when the arbitration bus, $\mathbf{AB[7..0]}^*$, becomes stable and equivalent to $\mathbf{cn[7.0]}$ on the input to a module and the indication within that module that it is the winner.

5.1.6 Arbitration states

The potential states that may be entered by a module involved in the arbitration process is illustrated in figure 7, along with the events that may result in a status change from each state. The figure shows examples of legal state transitions that may occur as a consequence of a control acquisition cycle.

Transitions from one state to another occur only when certain conditions are met.



NOTE—This figure is meant to provide a high-level overview, not a precise state diagram of the control acquisition process.

Figure 7 —Control acquisition states

5.1.7 Arbitration phases

Six phases of the three synchronization signals, $\mathbf{AP}^*$, $\mathbf{AQ}^*$, and $\mathbf{AR}^*$, are used during the control acquisition cycle. Normally, each phase is completed as quickly as propagation delays and metastability resolution times allow. If a two-pass arbitration competition is required, the entire control acquisition cycle is repeated.

The signals AP^* , AQ^* , and AR^* are synchronizing signals representing the wire-OR of the module signals ap^* , aq^* , and ar^* , respectively. For any of them to be released on the bus, each module must individually release its signal. The release of a signal is a global synchronization point for each module. No module may proceed to the next phase of the synchronization arbitration cycle until all modules have released the appropriate arbitration line. Similarly, the assertion of one of these signals indicates that a new phase is starting, and all modules, even if they are not participating in the current arbitration competition, must remain synchronized by asserting their own signal in response.

Figure 8 shows the sequence of phases and line state changes.

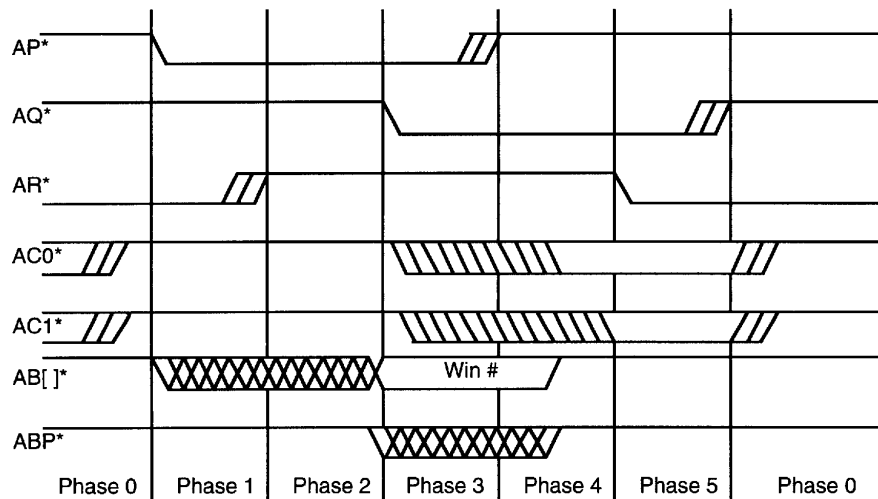


Figure 8 —Control acquisition sequence

A control acquisition cycle consists of six phases that proceed sequentially, starting with phase 0 and ending with phase 5. The beginning and end of each phase is defined by a transition on one of the arbitration synchronizing signals, AP^* , AQ^* , and AR^* . These signals ensure that all modules, whether competing or not, remain synchronized throughout the control acquisition cycle.

5.1.7.1 Phase 0: idle phase

Phase 0, the idle phase, is identified by AP^* released, AQ^* at logic zero, and AR^* asserted. This is the state of the arbitration synchronization lines when no competition cycle is in progress. It may be the midpoint between two competition cycles during a two-pass request or message. Modules that wish to send a message or request the bus assert ap^* . This is the first indication to other modules that a competition is beginning.

5.1.7.2 Phase 1: decision phase

Phase 1, the decision phase, is identified by AP^* asserted, AQ^* released, and AR^* asserted. Each module responds to the assertion of AP^* with the assertion of ap^* .

Since modules other than the initiator of the competition may decide to compete at the same time, the decision to compete is subject to metastability, due to the asynchronous relationship between the perceived assertion of AP^* and a module's internal message or bus request signal to the arbitration logic. Modules must resolve any metastable condition before the decision phase is terminated.

After each module has resolved any metastability and made its decision whether or not to compete during this arbitration cycle, it releases **ar***. Noncompeting modules are synchronized with the arbitration cycle and may not initiate a new competition until the sequence has ended with return to phase 0.

No module may release **ar*** before it has released **ac0*** and **ac1***, and, if it is a competitor, it has enabled its competition number, **cn[]**, and competition number parity, **cnp**, onto the arbitration bus, **ab[]***, and **abp***.

5.1.7.3 Phase 2: competition phase

Phase 2, the competition phase, is identified by **AP*** asserted, **AQ*** released, and **ARf** at logic zero. Upon detecting the beginning of phase 2, competitors begin timing the arbitration competition settling time.

At the beginning of phase 2, all modules start a 1 μ s to 2 μ s arbitration error timer. This timer notifies the system of an arbitration error if it expires before the assertion of **AQ***. This provides a recovery mechanism for some types of failures that would otherwise cause the bus to lock up.

Each competitor monitors its internal “win” signal from the arbitration competition logic. Because the competition is an iterative process, the state of this competitor signal may change several times before settling to its final value. If a module’s arbitration competition timer expires with its win signal asserted, it changes its state to winner (or master elect) and asserts **aq***.

If the 1 μ s arbitration error timer expires before the assertion of **AQ***, there has been an arbitration error. All modules detecting the error assert **aq*** and ensure that the arbitration competition terminates without a transfer of tenure by asserting both **ac0*** and **ac1***.

5.1.7.4 Phase 3: error check

Phase 3, the error check phase, is identified by **AP*** asserted, **AQ*** asserted, and **AR*** released. All modules, upon observing **AQ*** asserted, assert **aq***.

During phase 3, all modules latch the value of the winning competition number from the bus and check its parity. Competition losers may also check that the winning competition number is greater than their own. Any modules detecting an error condition assert **ac0*** and **ac1***.

In a distributed arbiter system, if the winner of the arbitration is making a bus request, that module becomes the master elect. Phase 3 continues as long as the master’s transaction on the parallel bus continues. Modules may request to depose the master elect by asserting **ac1*** during phase 3, causing the arbitration competition to terminate without a transfer of tenure. The completion of a two-pass arbitration competition takes precedence over an attempt to depose during the first pass. Note that the decision to depose the master elect is also subject to metastability.

In a distributed arbiter system, each module monitors **AS*** to detect its release by the master. This signals that the master is ending a bus transaction, and there may be a transfer of tenure. When **AS*** is released, each module releases **ap***.

NOTE — If **AS*** is asserted just as the bus enters phase 3, some modules may detect this condition, releasing **ap***, while others may not. Modules that have released **ap*** will be stuck, unable to depose the master elect, until the next release of **AS*** or assertion of **AC1***.

In a distributed arbiter system, modules sending arbitration messages signal that there is to be no transfer of tenure by asserting **ac1*** during phase 3.

In a central arbiter system, modules release **ap*** as soon as they have valid data on **ac0*** and **ac1***.

In either case, a module competing with a two-pass arbitration number and finding itself an intermediate winner asserts **ac1*** to prevent the premature transfer of message or tenure. Only when both passes have completed and there is a single winner is the message transmitted or tenure transferred. All modules also monitor **AC1***, and release **ap*** if they observe **AC1*** asserted. This indicates that the cycle has been canceled or is incomplete.

5.1.7.5 Phase 4: master release

Phase 4, the master release phase, is identified by **APf** at logic zero, **AQ*** asserted, and **AR*** released. At the beginning of phase 4 all modules release **ab[]***.

In a distributed arbiter system, all modules monitor **AC0*** and **AC1*** to determine when either an error is being reported, deposition is being requested, or tenure transfer is being inhibited.

In a distributed arbiter system, if the master wishes to perform additional transactions, it may cancel the transfer of tenure by asserting **ac1*** during phase 4. If it is ready to transfer tenure, and neither **AC0*** or **AC1*** are asserted, and **AKf** is at logic zero, it ensures that all information bus lines on the parallel bus are released. Then it asserts **ar***, signaling the transition to phase 5. If there is to be no transfer of tenure, the master may proceed immediately to phase 5 by asserting **ar***.

In a distributed arbiter system, each module, other than the master, starts a 1 μ s timer when **APf** is detected at logic zero. Then it monitors **AR***, awaiting its assertion by the master. On observing the assertion of **AR***, a module should disable the timer and assert **ar***. If the 1 μ s period expires due to some failure on the part of the master, the module should assert **ar*** to terminate phase 4.

In a central arbiter system, all modules immediately assert **ar*** to exit phase 4.

5.1.7.6 Phase 5: tenure/message transfer

Phase 5, the message or tenure transfer phase, is identified by **AP*** released, **AQ*** asserted, and **AR*** asserted. All modules save the state of **AC0*** and **AC1*** during phase 5, to determine if a message or tenure transfer is to take place, or whether it has been prevented by either cancellation or the detection of an arbitration error. Once they have latched this status, they release **aq***.

In a distributed arbiter system, the master elect, upon detecting the assertion of **AR*** and observing both **AC0*** and **AC1*** released, assumes tenure of the bus. If either is asserted, the current master retains ownership of the bus.

If **AC1*** was asserted at the beginning of phase 5, message or tenure transfer is inhibited. This may be for any of the following reasons:

- 1) An arbitration error occurred.
- 2) A second pass competition is required before the acquisition cycle can be completed.
- 3) In a central arbiter system, a second pass competition is required before the central arbitrated message can be completed.
- 4) In a distributed arbiter system, an arbitration message has been sent.
- 5) In a distributed arbiter system, the master has decided to retain the bus for additional transactions.
- 6) In a distributed arbiter system, the new master elect is being deposited by a higher priority module.

Following the first pass of a two-pass competition, any module that did not compete in the first acquisition cycle must become inhibited from competing during the second pass of the control acquisition cycle. Any competitor that was an intermediate winner must compete in the second pass using a competition number appropriate to the second pass of a two-pass arbitration competition.

After the second pass of the control acquisition cycle, any module that was inhibited at the start of that cycle may remove that inhibition, allowing it to be a competitor in the following arbitration competition.

5.1.8 Arbitration examples

5.1.8.1 Single-pass competition (central or distributed)

Figure 9 illustrates a single-pass arbitration competition which results in a transfer of tenure (in a distributed system) or general message transmission (in a central system). Note that the state of AS^* is irrelevant in a central arbiter system. The operation proceeds as follows:

- 1) When a distributed bus request or general arbitrated message request is made by one or more modules, those modules assert ap^* beginning phase 1.
- 2) Other modules, when they detect AP^* asserted, decide whether or not to compete and assert ap^* .
- 3) Competing modules activate their competition number on $ab[]^*$ and abp^* , and release ar^* .
- 4) After competitors detect ARf at logic zero, indicating the beginning of phase 2, they begin timing the arbitration competition settling time.
- 5) After the arbitration lines have settled, one of the modules detects it is the winner and asserts aq^* causing phase 3 to begin.
- 6) Modules then latch the identity of the winner from the $AB[]^*$ lines, check for errors, and in a distributed bus request, wait for the release of AS^* .
- 7) In a distributed bus request, when AS^* is released all modules release ap^* . In a general message request, ap^* is released immediately.
- 8) When modules detect APf at logic zero, phase 4 begins.
- 9) In a distributed bus request, when the master completes its final transaction, it asserts ar^* and changes its state to bystander. In a general message request, ar^* is asserted by all modules immediately.
- 10) In a distributed bus request, when the master elect detects AR^* asserted, indicating the beginning of phase 5, it changes its state to master and may begin transactions on the parallel bus.
- 11) Modules then detect that a transfer of tenure or message transmission occurred and release aq^* .

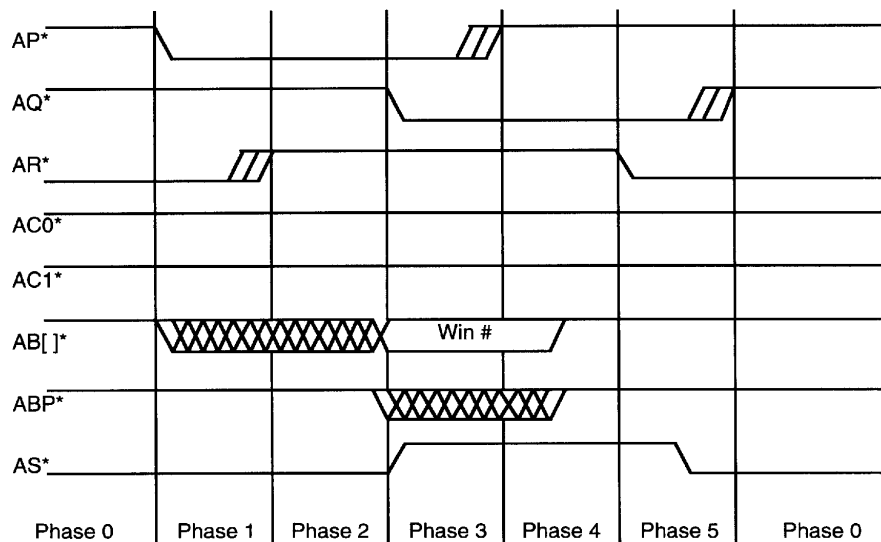


Figure 9 —Single-pass competition

5.1.8.2 Single-pass deposited (distributed) competition

Figure 10 illustrates a single-pass distributed bus request competition which is deposited, cancelling the transfer of tenure. The operation proceeds as follows:

- 1) When an arbitration request is made by one or more modules those modules assert **ap**^{*} beginning phase 1.
- 2) Other modules, when they detect **AP**^{*} asserted, decide whether or not to compete and assert **ap**^{*}.
- 3) Competing modules activate their competition number on **ab[]**^{*} and **abp**^{*}, and release **ar**^{*}.
- 4) After competitors detect **ARf** at logic zero, indicating the beginning of phase 2, they begin timing the arbitration competition settling time.
- 5) After the arbitration lines have settled one of the modules detects it is the winner and asserts **aq**^{*} causing phase 3 to begin.
- 6) Modules then latch the identity of the master elect from the **AB[]**^{*} lines, check for errors, and wait for the release of **AS**^{*}.
- 7) A noncompeting module then receives a bus request and determines that its competition number would have won the previous competition. It therefore asserts **ac1**^{*} to depose the master elect and releases **ap**^{*}.
- 8) Other modules upon detecting **AC1**^{*} asserted release **ap**^{*}.
- 9) When modules detect **AC1**^{*} asserted and **APf** at logic zero, they assert **ar**^{*} to begin phase 5.
- 10) Modules then detect a transfer of tenure did not occur and release **aq**^{*}.
- 11) When modules detect **AQf** at logic zero, they begin a new arbitration competition.

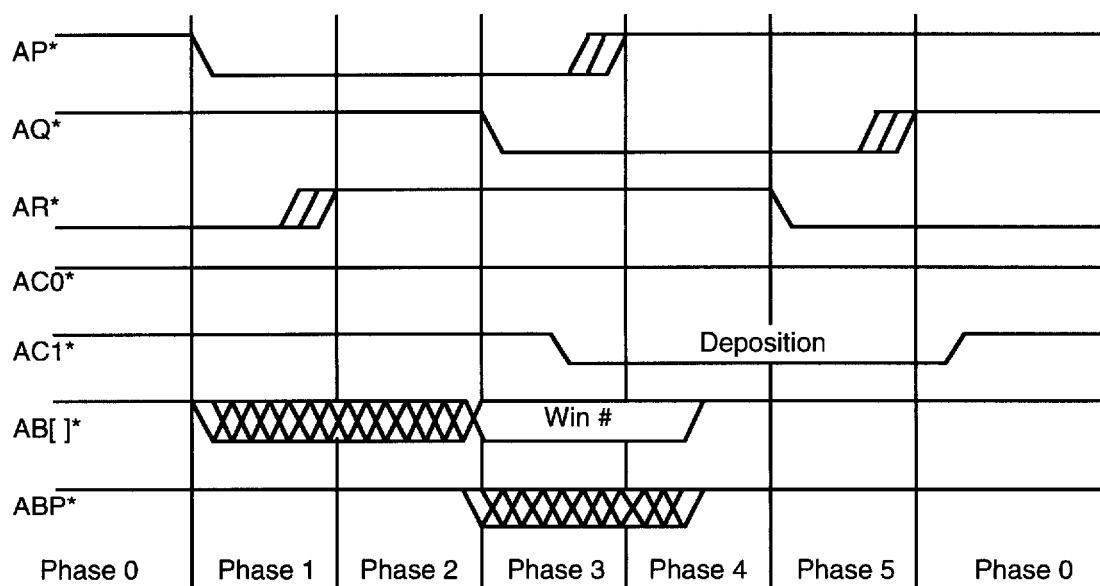


Figure 10 —Single-pass deposited competition

5.1.8.3 Distributed arbitration message

Figure 11 illustrates a two-pass distributed arbitration competition resulting in an distributed arbitration message being sent. The operation proceeds as follows:

- 1) When an arbitration message request is made by one or more modules those modules assert **ap**^{*} beginning phase 1.
- 2) Other modules, when they detect **AP**^{*} asserted, decide whether or not to compete and assert **ap**^{*}.
- 3) Competing modules activate their competition number on **ab[]**^{*} and **abp**^{*}, and release **ar**^{*}.
- 4) After competitors detect **ARf** at logic zero, indicating the beginning of phase 2, they begin timing the arbitration competition settling time.
- 5) The modules sending arbitration messages quickly determine they are the winner and assert **aq**^{*} causing phase 3 to begin.

- 6) The modules sending an arbitration message assert **ac1*** to cause a second pass competition to occur and release **ap***.
- 7) Other modules upon detecting **AC1*** asserted release **ap***.
- 8) When modules detect **AC1*** asserted and **APf** at logic zero they assert **ar*** to begin phase 5.
- 9) Modules then detect a transfer of tenure did not occur and that a second pass is required and release **aq***.
- 10) When modules that are not inhibited by losing the first pass of the competition detect **AQf** at logic zero, they begin a new arbitration competition.
- 11) The competition then proceeds with the module with the highest arbitration message number winning. The competition is again cancelled to prevent a transfer of tenure.

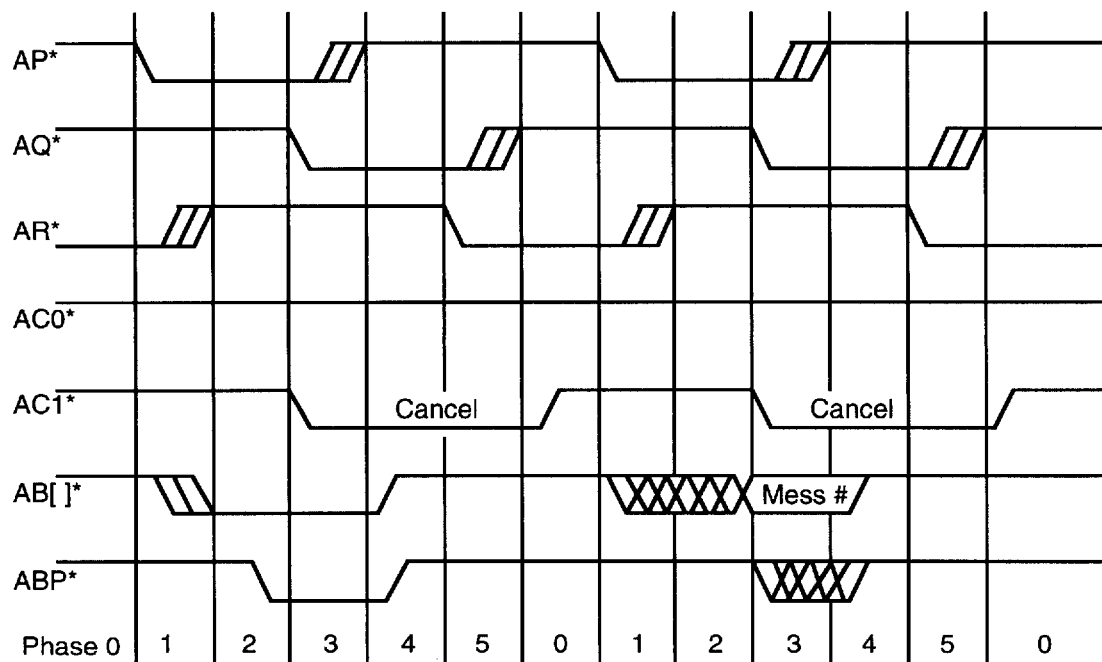


Figure 11 —Arbitration message

5.1.8.4 Two-pass competition (central or distributed)

Figure 12 illustrates a two-pass arbitration competition that results in a transfer of tenure (in a distributed system) or a central message transmission (in a central system). The first pass consists of a single-pass competition which is canceled by the first-pass winners. Competitors losing the first pass are inhibited from competing in the second pass. The second pass operates identically to a single-pass competition with transfer of tenure as was described in 5.1.8.1 except that only first-pass winners compete. Note that the state of **AS*** is again irrelevant in a central arbiter system.

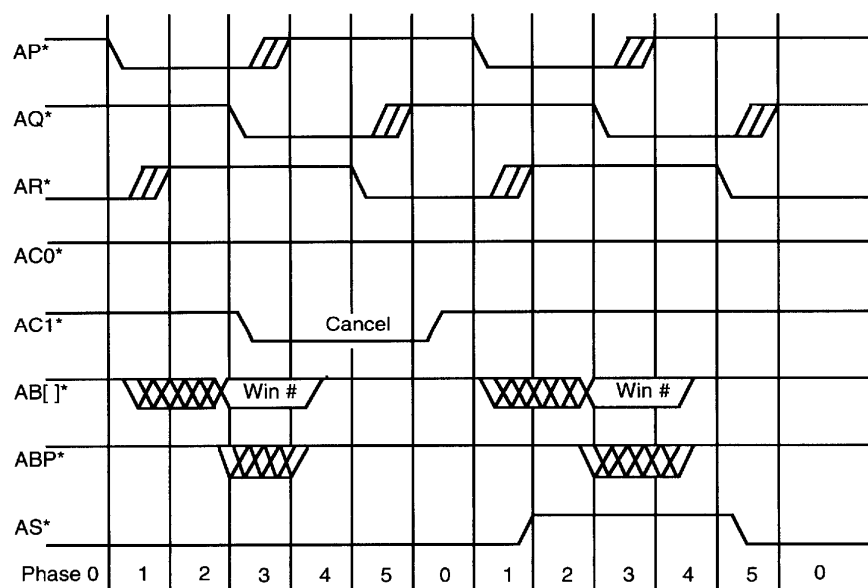


Figure 12 —Two-pass competition

5.1.8.5 Single-pass competition with error

Figure 13 illustrates a single-pass arbitration competition in which an error is detected cancelling the transfer of tenure or transmission of the message. The operation proceeds normally until the error is detected in phase 3 causing one or more modules to assert both $ac0^*$ and $ac1^*$. The operation then proceeds in the same way as other canceled cycles except that modules take note of the error in phase 5 and take the appropriate error recovery action.

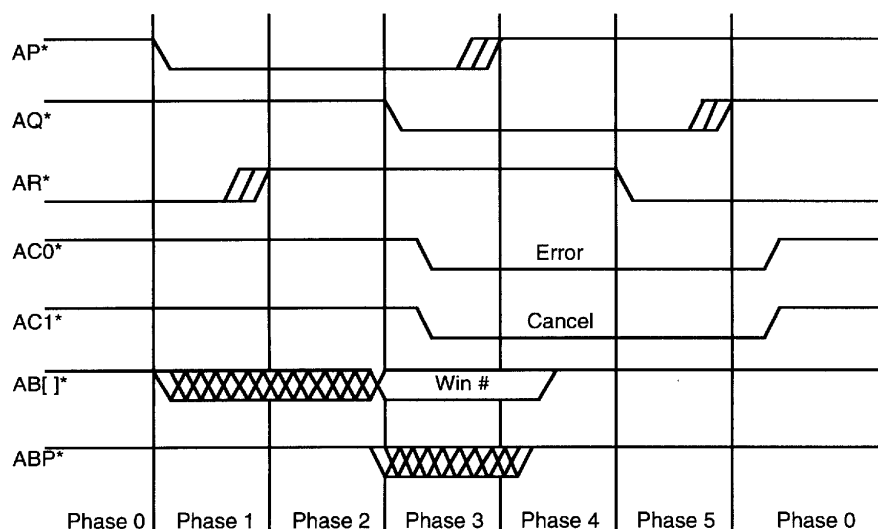


Figure 13 —Single-pass competition with error

5.2 Specification

5.2.1 Arbitrated message attributes—central arbiter

ARBITRATED_MESSAGE_REQUEST. Modules may set ARBITRATED_MESSAGE_REQUEST when CENTRAL_ARBITER & ~INIT & DISTRIBUTED_MESSAGE_ENABLE & ~ARBITRATED_MESSAGE_SENT & ~CENTRAL_MESSAGE_REQUEST & ~SECOND_PASS & PHASE_0 & ~**ap*** to send an arbitration message. Modules shall clear ARBITRATED_MESSAGE_REQUEST when INIT | (~CENTRAL_ARBITER | ARBITRATED_MESSAGE_SENT | ~DISTRIBUTED_MESSAGE_ENABLE) & ~SECOND_PASS & PHASE_0 & ~**ap***.

NOTE — The decision to set ARBITRATED_MESSAGE_REQUEST is subject to metastability. Modules shall not release **ar*** until adequate time has been allowed for the resolution of this condition.

CENTRAL_MESSAGE_REQUEST. Modules may set CENTRAL_MESSAGE_REQUEST when CENTRAL_ARBITER & ~INIT & DISTRIBUTED_MESSAGE_ENABLE & ~CENTRAL_MESSAGE_SENT & ~ARBITRATED_MESSAGE_REQUEST & ~SECOND_PASS & PHASE_0 & ~**ap*** to send an arbitration message. Modules shall clear CENTRAL_MESSAGE_REQUEST when ~CENTRAL_ARBITER | INIT | (CENTRAL_MESSAGE_SENT | ~DISTRIBUTED_MESSAGE_ENABLE) & ~SECOND_PASS & PHASE_0 & ~**ap***.

NOTE — The decision to set CENTRAL_MESSAGE_REQUEST is subject to metastability. Modules shall not release **ar*** until adequate time has been allowed for the resolution of this condition.

ARBITRATED_MESSAGE_SENT. A module shall set ARBITRATED_MESSAGE_SENT when ARBITRATED_MESSAGE_REQUEST & WINNER & ~**AC0*** & PHASE_5. A module shall clear ARBITRATED_MESSAGE_SENT when ~ARBITRATED_MESSAGE_REQUEST.

CENTRAL_MESSAGE_SENT. A module shall set CENTRAL_MESSAGE_SENT when CENTRAL_MESSAGE_REQUEST & WINNER & ~SECOND_PASS_REQUIRED & ~**AC0*** & PHASE_5. A module shall clear CENTRAL_MESSAGE_SENT when ~CENTRAL_MESSAGE_REQUEST.

PR[7..0]. A module may load an 8-bit priority value into **PR[7..0]** to change central arbiter priority levels when ~CENTRAL_MESSAGE_REQUEST.

RQ. Modules may set **RQ** to change the priority level of central arbiter request line **RQ1***. Modules may clear **RQ** to change the priority level of central arbiter request line **RQ0***. Any change to **RQ** shall only occur when ~CENTRAL_MESSAGE_REQUEST.

AM[13..0]. A module shall load a 7-bit arbitrated message value into **AM[6..0]** when (PHASE_0 | PHASE_1) & ARBITRATED_MESSAGE_REQUEST & **ar***. A module shall load a 14-bit arbitrated message value into **AM[13..0]** when ~SECOND_PASS & (PHASE_0 | PHASE_1) & CENTRAL_MESSAGE_REQUEST & **ar***, where **AM[13..6]** shall equal **PR[7..0]**, **AM5** shall equal **RQ**, and **AM[4..0]** shall equal **GA[4..0]**.

5.2.2 Arbitrated message attributes—distributed arbiter

DISTRIBUTED_MESSAGE_REQUEST. Modules may set DISTRIBUTED_MESSAGE_REQUEST when ~CENTRAL_ARBITER & ~INIT & DISTRIBUTED_MESSAGE_ENABLE & ~DISTRIBUTED_ARBITER_REQUEST & ~DISTRIBUTED_MESSAGE_SENT & ~SECOND_PASS & PHASE_0 & ~**ap*** to send an arbitration message. Modules shall clear

DISTRIBUTED_MESSAGE_REQUEST when CENTRAL_ARBITER | INIT |
(DISTRIBUTED_MESSAGE_SENT | ~DISTRIBUTED_MESSAGE_ENABLE) & ~SECOND_PASS &
PHASE_0 & ~ap*.

NOTE — The decision to set DISTRIBUTED_MESSAGE_REQUEST is subject to metastability. Modules shall not release ar* until adequate time has been allowed for the resolution of this condition.

DISTRIBUTED_MESSAGE_SENT. A module shall set DISTRIBUTED_MESSAGE_SENT when DISTRIBUTED_MESSAGE_REQUEST & WINNER & ~SECOND_PASS_REQUIRED & ~AC0* & PHASE_5. A module shall clear DISTRIBUTED_MESSAGE_SENT when ~DISTRIBUTED_MESSAGE_REQUEST.

DISTRIBUTED_MESSAGE. A module shall set DISTRIBUTED_MESSAGE if ~CENTRAL_ARBITER & PHASE_3 & ~SECOND_PASS & AB7* & AB6* & AB5* & AB4* & AB3* & AB2* & AB1* & AB0*. Modules shall maintain DISTRIBUTED_MESSAGE set until INIT | PHASE_0 & ~SECOND_PASS.”

AM[6..0]. A module shall load an 7-bit arbitration message value into AM[6..0] when (phase_0 1 phase_1) & DISTRIBUTED_MESSAGE_request & ar*.

5.2.3 Arbitration attributes—distributed arbiter

DISTRIBUTED_ARBITER_REQUEST. Modules may set DISTRIBUTED_ARBITER_REQUEST when ~CENTRAL_ARBITER & ~INIT & MASTER_ENABLE & ~MASTER & ~DISTRIBUTED_MESSAGE_REQUEST & ~SECOND_PASS & PHASE_0 & ~ap* to obtain bus mastership. Modules shall clear DISTRIBUTED_ARBITER_REQUEST when CENTRAL_ARBITER | INIT | (MASTER_ENABLE | DISTRIBUTED_MASTER) & ~SECOND_PASS & PHASE_0 & ~ap*.

NOTE — The decision to set DISTRIBUTED_ARBITER_REQUEST is subject to metastability. Modules shall not release ar* until adequate time has been allowed for the resolution of this condition.

DEPOSITION. A module may set DEPOSITION if ~CENTRAL_ARBITER & ~INIT & MASTER_ENABLE & ~BUS_HOLD & PHASE_3 & ap* if it needs to send an arbitration message or if it needs to obtain bus mastership and will use a higher arbitration number (PR[7..0], RR, GA[4..0]) than that of the current winner. Modules shall maintain DEPOSITION set until PHASE_0.

NOTE — The decision to set DEPOSITION is subject to metastability. Modules shall not release ap* until adequate time has been allowed for the resolution of this condition.

NO_ARBITRATION_OWNER. Modules shall set no_ARBITRATION_OWNER when init and shall maintain it set until phase_5 & ~AC1.

ARBITRATION_OWNER. A module shall set ARBITRATION_OWNER if ~INIT & (DISTRIBUTED_ARBITER_REQUEST & WINNER & ~SECOND_PASS_REQUIRED & ~AC1* & PHASE_5 and maintain it set until INIT | (~DISTRIBUTED_ARBITER_REQUEST & ~SECOND_PASS_REQUIRED & ~AC1* | DISTRIBUTED_ARBITER_REQUEST & ~WINNER & ~SECOND_PASS_REQUIRED & ~AC* & PHASE_5.

DISTRIBUTED_GRANT. A module shall set DISTRIBUTED_GRANT when DISTRIBUTED_ARBITER_REQUEST & (WINNER & ~SECOND_PASS_REQUIRED & ~AC1* & PHASE_5 | ARBITRATION_OWNER). A module shall clear DISTRIBUTED_GRANT when ~DISTRIBUTED_ARBITER_REQUEST.

DISTRIBUTED_MASTER. A module shall set DISTRIBUTED_MASTER when DISTRIBUTED_GRANT & ~ET*. A module shall clear DISTRIBUTED_MASTER when ~DISTRIBUTED_ARBITER_REQUEST & ~et*.

TWO_PASS. A module shall set TWO_PASS when $\sim(\text{PR7} \ \& \ \text{PR6} \ \& \ \text{PR5} \ \& \ \text{PR4} \ \& \ \text{PR3} \ \& \ \text{PR2} \ \& \ \text{PR1})$.

PR[7..0]. A module shall load an 8-bit priority value into **PR[7..0]** when $\sim\text{SECOND_PASS} \ \& \ (\text{PHASE_0} \mid \text{PHASE_1}) \ \& \ \text{DISTRIBUTED_ARBITER_REQUEST} \ \& \ \text{ar}^*$.

ROUND_ROBIN. Modules shall either maintain a ROUND_ROBIN attribute for each priority class or if multiple priority classes are associated with a single ROUND_ROBIN attribute shall clear ROUND_ROBIN each time they changes the priority class associated with the ROUND_ROBIN attribute. Priority Class is defined to be a set of competition numbers that have identical priority fields and are discriminated by the Round Robin and Geographical Address fields.

A module shall set ROUND_ROBIN for a priority class if $\sim\text{INIT} \ \& \ \text{UNIQUENESS_GREATER} \ \& \ \text{PHASE_5} \ \& \ \sim\text{AC1}^*$ and shall maintain it set until $\text{INIT} \mid \sim\text{UNIQUENESS_GREATER} \ \& \ \text{PHASE_5} \ \& \ \sim\text{AC1}^*$. A module may maintain ROUND_ROBIN set if $\text{BUSY_STATUS} \mid \text{WAIT_STATUS}$ is received. A module may set ROUND_ROBIN if it has relinquished bus tenure early to a module of higher priority class.

GA[4..0]. A 5-bit value, **GA[4..0]**, shall be set to **GA[4..0]**^{*}.

UNIQUENESS_GREATER. Modules shall set UNIQUENESS_GREATER if $\text{PHASE_3} \ \& \ \sim\text{DISTRIBUTED_MESSAGE} \ \& \ \text{AB7}^*$ and the value on **AB[4..0]**^{*} is greater than **GA[4..0]** and its priority level is the same as that of the current winner. Modules shall maintain UNIQUENESS_GREATER until PHASE_0 .

5.2.4 General arbitration and message attributes

COMPETITOR. A module shall set COMPETITOR if $\sim\text{INIT} \ \& \ \sim\text{SECOND_PASS} \ \& \ (\text{ARBITRATED_MESSAGE_REQUEST} \ \& \ \sim\text{ARBITRATED_MESSAGE_SENT} \mid \text{CENTRAL_MESSAGE_REQUEST} \ \& \ \sim\text{CENTRAL_MESSAGE_SENT} \mid \text{DISTRIBUTED_ARBITER_REQUEST} \ \& \ \sim\text{ARBITRATION_OWNER} \mid \text{DISTRIBUTED_MESSAGE_REQUEST} \ \& \ \sim\text{DISTRIBUTED_MESSAGE_SENT}) \ \& \ (\text{PHASE_0} \mid \text{PHASE_1}) \ \& \ \text{ar}^*$ and shall maintain it set until $\text{INIT} \mid (\text{AC0}^* \mid \sim\text{SECOND_PASS_REQUIRED} \mid \sim\text{WINNER}) \ \& \ \text{PHASE_5}$.

CANCEL. Modules set CANCEL if $\text{phase_3} \ \& \ (\text{DEPOSITION} \mid \text{SECOND_PASS_REQUIRED} \ \& \ \text{COMPETITOR} \mid \text{ARBITRATION_ERROR} \mid \text{DISTRIBUTED_MESSAGE}) \mid \text{PHASE_4} \ \& \ \text{ARBITRATION_OWNER} \ \& \ \text{TRANSACTION_BEGIN}$. Modules shall clear CANCEL if PHASE_0 .

CN7. A module shall set **cn7** when $\text{ARBITRATED_MESSAGE_REQUEST} \mid \text{CENTRAL_MESSAGE_REQUEST} \ \& \ \text{SECOND_PASS} \mid \text{DISTRIBUTED_ARBITER_REQUEST} \ \& \ (\sim\text{TWO_PASS} \mid \text{SECOND_PASS}) \mid \text{DISTRIBUTED_MESSAGE_REQUEST}$.

CN6. A module shall set **cn6** when $\text{ARBITRATED_MESSAGE_REQUEST} \ \& \ \text{AM6} \mid \text{CENTRAL_MESSAGE_REQUEST} \ \& \ (\sim\text{SECOND_PASS} \ \& \ \text{AM13} \mid \text{SECOND_PASS} \ \& \ \text{AM6}) \mid \text{DISTRIBUTED_ARBITER_REQUEST} \ \& \ ((\sim\text{TWO_PASS} \mid \text{SECOND_PASS}) \ \& \ \text{PRO} \mid \text{TWO_PASS} \ \& \ \sim\text{SECOND_PASS} \ \& \ \text{PR7}) \mid \text{DISTRIBUTED_MESSAGE_REQUEST} \ \& \ (\sim\text{SECOND_PASS} \mid \text{SECOND_PASS} \ \& \ \text{AM6})$.

CN5. A module shall set **cn5** when $\text{ARBITRATED_MESSAGE_REQUEST} \ \& \ \text{AM5} \mid \text{CENTRAL_MESSAGE_REQUEST} \ \& \ (\sim\text{SECOND_PASS} \ \& \ \text{AM12} \mid \text{SECOND_PASS} \ \& \ \text{AM5}) \mid \text{DISTRIBUTED_ARBITER_REQUEST} \ \& \ ((\sim\text{TWO_PASS} \mid \text{SECOND_PASS}) \ \& \ \text{ROUND_ROBIN} \mid \text{TWO_PASS} \ \& \ \sim\text{SECOND_PASS} \ \& \ \text{PR6}) \mid \text{DISTRIBUTED_MESSAGE_REQUEST} \ \& \ (\sim\text{SECOND_PASS} \mid \text{SECOND_PASS} \ \& \ \text{AM5})$.

CN4. A module shall set **cn4** when $\text{ARBITRATED_MESSAGE_REQUEST} \ \& \ \text{AM4} \mid \text{CENTRAL_MESSAGE_REQUEST} \ \& \ (\sim\text{SECOND_PASS} \ \& \ \text{AM11} \mid \text{SECOND_PASS} \ \& \ \text{AM4}) \mid \text{DISTRIBUTED_ARBITER_REQUEST} \ \& \ ((\sim\text{TWO_PASS} \mid \text{SECOND_PASS}) \ \& \ \text{GA4} \mid \text{TWO_PASS} \ \& \ \sim\text{SECOND_PASS} \ \& \ \text{GA4})$.