

Recognized as an
American National Standard (ANSI)

IEEE Std 1014™-1987(R2008)

IEEE Standard for A Versatile Backplane Bus: VMEbus

Technical Committee on Microprocessors and Microcomputers
of the
IEEE Computer Society

Reaffirmed 31 January 2008

Approved 12 March 1987

IEEE-SA Standards Board

Approved 11 September 1987

American National Standards Institute

Abstract: This IEEE 1014 standard specifies a high-performance backplane bus for use in microcomputer systems that employ single or multiple microprocessors. It is based on the VMEbus specification, released by the VME Manufacturers' Group in August 1982. The bus includes four subbuses: data transfer bus, priority interrupt bus, arbitration bus, and utility bus. The data transfer bus supports 8-, 16-, and 32-bit transfers over a non-multiplexed 32-bit data and address highway. The transfer protocols are asynchronous and fully handshaken. The priority interrupt bus provides real-time interrupt services to the system. The allocation of bus mastership is performed by the arbitration bus, which allows to implement round robin and prioritized arbitration algorithms. The utility bus provides the system with power-up and power-down synchronization. The mechanical specifications of boards, backplanes, subracks, and enclosures are based on IEC 297 specification, also known as the Euroboard form factor.

345 East 47th Street, New York, NY 10017, USA

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.

Copyright © 1988 by **The Institute of Electrical and Electronics Engineers, Inc.**

ISBN 0-471-61601-X

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

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

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

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

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

Secretary, IEEE Standards Board
345 East 47th Street
New York, NY 10017
USA

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

Foreword

(This Foreword is not a part of ANSI/IEEE Std 1014-1987, IEEE Standard for A Versatile Backplane Bus: VMEbus.)

The architectural concepts of the VMEbus are based on the VERSAbus, which was developed in the late 1970's by Motorola. Motorola's European Microsystems group in Munich, West Germany proposed the development of a VERSAbus-like product line based on the Eurocard mechanical standard. To demonstrate the concept, Max Loesel and Sven Rau developed three prototype boards: a 68000 CPU board, a dynamic memory board, and a static memory board. They named the new bus VERSAbus-E. It was the VERSAbus-E, renamed VMEbus, that Motorola, Mostek, and Signetics have agreed to jointly develop and support.

John Black of Motorola, Craig MacKenna of Mostek, and Cecil Kaplinsky of Signetics developed the first draft of the VMEbus specification. In October of 1981, at the *System 81* show in Munich, West Germany, Motorola, Mostek, and Signetics announced their joint support of the VMEbus, and placed Revision A of the specification in the public domain.

In August of 1982, Revision B of the VMEbus specification was published by the newly formed VMEbus Manufacturers' Group. This new revision refined the electrical specifications for the signal line drivers and receivers, and also brought the mechanical specifications more in line with the developing IEC 297-3 standard, the formal specifications of the Eurocard mechanical formats.

In the later part of 1982, the French delegation to the International Electrotechnical Commission (IEC) proposed Revision B of the VMEbus as an international standard. The IEC SC47B subcommittee nominated Mira Pauker of Philips, France, as the chairperson of an editorial committee, formally starting international standardization of the VMEbus.

In March of 1983, the IEEE Microprocessor Standards Committee (MSC) requested authorization to establish a working group to standardize the VMEbus. This request was approved by the IEEE Standards Board, and the P1014 Working Group was established. Wayne Fischer was appointed first chairman of the working group. John Black served as chairman of the P1014 Technical Sub-committee.

The IEC and the IEEE distributed copies of Revision B for comment, and both received requests for change. Additional requests for change were received from the members of the VMEbus Manufacturers' Group. These comments made it clear that it was time to go onward past Revision B. In December of 1983, a meeting was held that included John Black, Mira Pauker, Wayne Fischer, and Craig MacKenna. It was agreed that a Revision C should be created, and that it should take into consideration all the comments received by the three organizations. John Black and Shlomo Pri-Tal of Motorola incorporated the changes from all sources into a common document. The Manufacturers' Group called it Revision C. 1, the IEEE called it P1014 draft 1.2, and the IEC called it the IEC 821 BUS. Subsequent ballots in the IEEE P1014 Working Group and in the MSC resulted in more comments, and required that the IEEE P1014 draft be updated. This work resulted in draft IEEE P1014 .

The process that led to the development of ANSI/IEEE Std 1014 embodies the philosophy that the “...*IEEE will cooperate with standardizing groups throughout the world in the preparation of standards...*” as expressed in the IEEE Standards Manual.

The development of the VMEbus was a team effort, which involved experts from the United States, Britain, West Germany, France, and many other countries. It is truly an internationally developed standard. However, the contribution of several individuals is worthy of special recognition:

John Black and Craig MacKenna were the key individuals in the creation of the VMEbus specification. Their efforts, expertise, and perseverance have guided the development of the VMEbus to its present form.

Max Loesel and Sven Rau are recognized for proposing and demonstrating the feasibility of a Eurocard-based 32-bit backplane bus.

Eike Waltz contributed extensively to the mechanical specifications.

Wayne Fischer, the first Chairman, guided the IEEE P1014 Working Group during its first three years.

Mira Pauker, Arlan Harris, and Shlomo Pri-Tal contributed to the development of Revision C of the specification.

Also worthy of recognition are Paul Borrill who contributed to the electrical specifications, Ken Smith who contributed to the mechanical specifications, and T. J. Chaney who was instrumental in preparing Appendix D.

And finally, special thanks to Tom Leonard who assisted Shlomo Pri-Tal as Vice Chairman of the P1014 Working Group.

The IEEE P1014 Technical Subcommittee had the following membership:

John Black, *Chair*
Craig MacKenna, *Secretary*

Thomas Harkaway
Arlan Harris

Doug Kraft

Thomas Leonard
Shlomo Pri-Tal

The IEEE P1014 Working Group had the following membership:

Shlomo Pri-Tal, *Chair*
Thomas Leonard, *Vice Chair*

Philip E. Abbate
Stephen Ades
Robert L. Ayers
Don Baril
Richard F. Barry
Drew Berding
I. M. Bisset
John Black
Neil Borkowicz
Paul Borrill
Steven Brandon
Duncan Campbell
Raymond W. Chateau
K. R. Clohessy
Richard M. DeBock
Steve Deiss
Glen Diestelhorst
Gary Dool
Ian R. Duncan
Tim J. Elsmore
Ken Finster
Wayne Fischer
Gene Freehauf

Ken Goertzen
Jim Green
Thomas Harkaway
Arlan Harris
Bill Holloway
Tom Hunter
Jurgen Jakel
Hugh Johnson
Cecil Kaplinsky
Hans Karlsson
Doug Kraft
Patrick E. Lannan
D.C. Liddell
Craig MacKenna
Rick Main
Andy McMillan
Lance McNally
Frank Melanson
Jeffrey R. Millar
Joseph Ng
L. Parker
Mira Pauker
Robert Phillips
Robert Pogson

Sven Rau
Jack Regula
Kim Rubin
Norman Schneidewind
George Schreck
Ron Schreck
Craig Scott
Bill Shields
Ken Smith
Basil Smith
Larry Sollman
Bob Squirrell
Michael Thompson
Michael UnTerweger
Gregory S. Urban
S. P. Verma
Eike Waltz
David Weller
John Wemekamp
Bryant West
C. J. White
William D. Winget
Ron Wolfe

The following members of the Microprocessor Standards Committee voted in the ballot that approved the IEEE P1014 for sponsor ballot by the Technical Committee on Microprocessor and Microcomputer (TCMM):

Bob Davis, Chair
Clyde Camp, Secretary

Matt Biewer
Richard Boberg
Paul Borrill
Steve Cooper
Wayne Fischer
Gordon Force
Martin Freeman
David Gustavson

Tom Harkaway
David James
Laurel Kaleda
Richard Karpinski
Doug Kraft
Patrick Laprocina
Gerry Laws
Tom Leonard
Gary Lyons

Craig MacKenna
Deene Ogden
Shlomo Pri-Tal
Michael Smolin
Robert Stewart
Eike Waltz
George White
Fritz Whitting

The members of the Executive Committee of the Technical Committee on Microprocessors and Microcomputers were as follows:

Martin Freeman, Chair

Bob Davis
Jim Flournoy

Glen Langdon

Michael Smolin
Bob Stewart

The following members of the Technical Committee on Microprocessors and Microcomputers were on the balloting body that approved this document for submission to the IEEE Standards Board:

Andrew Allison
J. Ashenden
Geoff Baldwin
Matt Biewer
John Black
Richard Boberg
Paul Borrill
Bradley Brown
Clyde Camp
John D. Charlton
William Cody
Steve Cooper
Tim Davey
Bob Davis
Randy Davis
Shirish P. Deodhar

Steve Diamond
Wayne Fischer
Jim Flournoy
Gordon Force
Martin Freeman
S. Ganesan
D. Gustavson
Thomas Harkaway
Richard James
David James
Laurel Kaleda
Richard Karpinski
Doug Kraft
Glen Langdon
Gerry Laws
Tom Leonard

Gary Lyons
Rae McLellan
Jim Mooney
Gary Nelson
Deene Ogden
Shlomo Pri-Tal
Richard Rawson
Mike Smolin
Michael Teener
Eike Waltz
Carl Warren
George White
Fritz Whittington
Thomas Wicklund
Andrew Wilson
Anthony Winter

When the IEEE Standards Board approved this standard on June 11, 1987, it had the following membership:

Donald C. Fleckenstein, *Chair*
Marco W. Migliaro, *Vice Chair*
Andrew G. Salem, *Secretary*

James H. Beall
Dennis Bodson
Marshall L. Cain
James M. Daly
Stephen R. Dillon
Eugene P. Fogarty
Jay Forster
Kenneth D. Hendrix
Irvin N. Howell

Leslie R. Kerr
Jack Kinn
Irving Kolodny
Joseph L. Koepfinger^{*}
Edward Lohse
John May
Lawrence V. McCall
L. Bruce McClung
Donald T. Michael^{*}

L. John Rankine
John P. Riganati
Gary S. Robinson
Frank L. Rose
Robert E. Rountree
Sava I. Sherr^{*}
William R. Tackaberry
William B. Wilkens
Helen M. Wood

^{*}Member emeritus

CLAUSE	PAGE
1. Introduction	1
1.1 Objectives.....	1
1.2 Interface System Elements	1
1.3 Specification Diagrams	6
1.4 Terminology	6
1.5 Protocol	8
1.6 System Examples and Explanations	10
2. Data Transfer Bus	10
2.1 Introduction.....	10
2.2 Data-Transfer-Bus Lines.....	11
2.3 DTB Modules — Basic Description	18
2.4 Typical Operation	36
2.5 Data-Transfer-Bus Acquisition	43
2.6 DTB Timing Rules and Observations	45
3. DTB Arbitration Bus.....	81
3.1 Introduction.....	81
3.2 Arbitration Bus Lines.....	83
3.3 Functional Modules.....	85
3.4 Typical Operation	92
3.5 Race Conditions Between Master Requests and Arbiter Grants.....	99
4. Priority Interrupt Bus	100
4.1 Introduction.....	100
4.2 Priority Interrupt Bus lines.....	103
4.3 Priority Interrupt Bus Modules — Basic Description.....	106
4.4 Typical Operation	116
4.5 Race Conditions	122
4.6 Priority Interrupt Bus Timing Rules and Observations	123
5. Utility Bus	142
5.1 Introduction.....	142
5.2 Utility Bus Signal Lines.....	142
5.3 Utility Bus Modules	142
5.4 System Initialization and Diagnostics.....	146
5.5 Power Pins.....	149
5.6 Reserved Line	150
6. Electrical Specifications.....	151
6.1 Introduction.....	151
6.2 Power Distribution	151
6.3 Electrical Signal Characteristics	152
6.4 Bus Driving and Receiving Requirements.....	153
6.5 Backplane Signal Line Interconnections.....	159
6.6 User-Defined Signals	165

CLAUSE	PAGE
6.7 Signal Line Drivers and Terminations	165
7. Mechanical Specifications	167
7.1 Introduction	167
7.2 Boards	168
7.3 Front Panels.....	176
7.4 Backplanes	185
7.5 Assembly of Subracks.....	192
7.6 Backplane Connectors and Board Connectors.....	195
Annex A Glossary (Informative)	197
Annex B Signal Line Description (Informative).....	201
Annex C Use of the SERCLK and SERDAT* Lines (Informative).....	204
Annex D Metastability and Synchronization (Informative).....	206
Annex E Permissible Capability Subsets (Informative).....	217

An American National Standard

IEEE Standard for A Versatile Backplane Bus: VMEbus

1. Introduction

1.1 Objectives

This standard defines an interfacing system used to interconnect data processing, data storage, and peripheral control devices in a tightly coupled hardware configuration. The system has the following objectives:

- 1) To allow communication between devices on the bus without disturbing the internal activities of other devices interfaced to the bus.
- 2) To specify the electrical and mechanical system characteristics required to design devices that will reliably and unambiguously communicate with other devices interfaced to the bus.
- 3) To specify protocols that precisely define the interaction between the bus and devices interfaced to it.
- 4) To provide terminology and definitions that describe system protocols.
- 5) To allow a broad range of design latitude so that the designer can optimize cost or performance, or both, without affecting the system compatibility.
- 6) To provide a system where performance is primarily device limited, rather than system interface limited.

1.2 Interface System Elements

1.2.1 Basic Definitions

The structure of the system can be described from its mechanical structure and its functional structure. The mechanical specification describes the physical dimensions of subracks, backplanes, front panels, plug-in boards, etc. The functional specification describes the manner in which the bus operates, the functional modules that are involved in each transaction, and the rules that govern their behavior. This section provides informal definitions for some basic terms used to describe the physical and the mechanical structure.

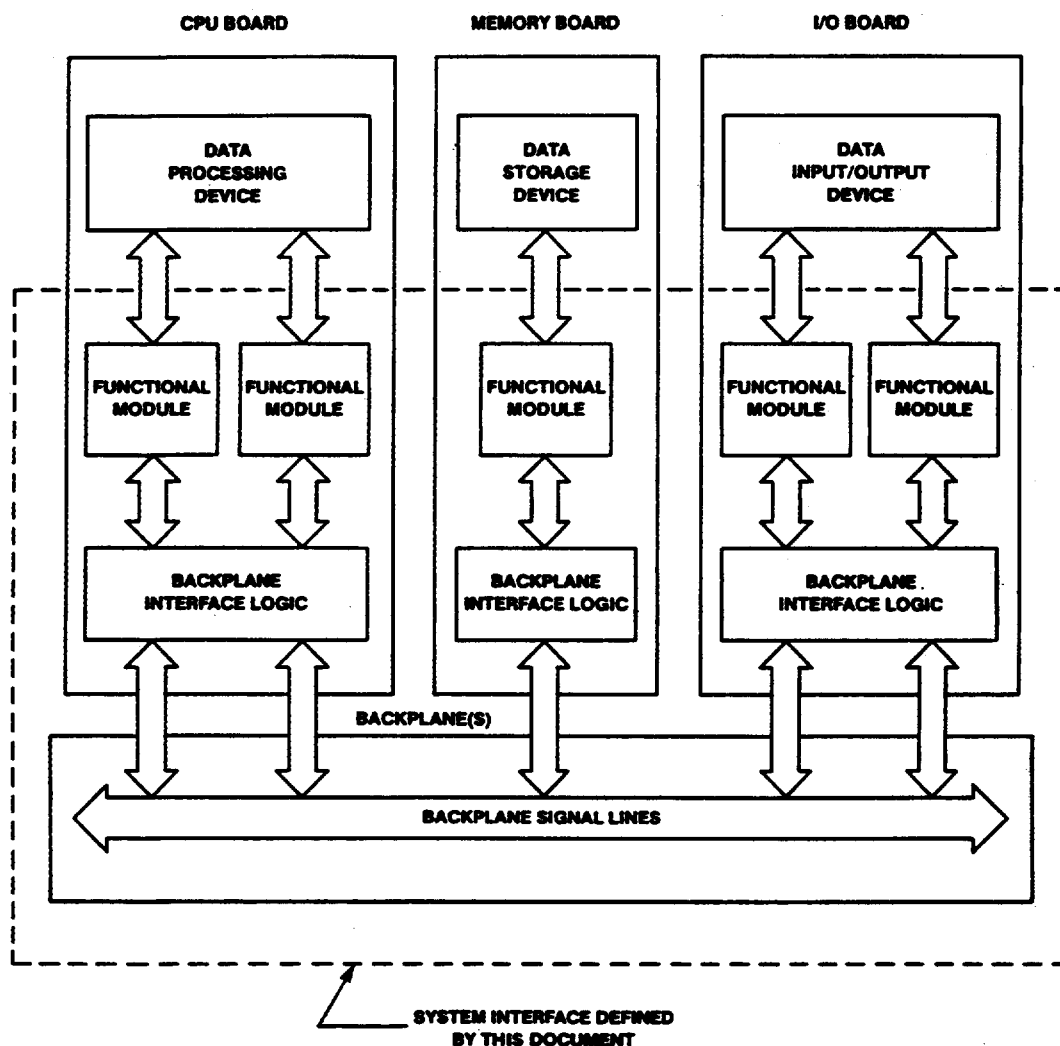


Figure 1—System Elements Defined by the Standard

1.2.1.1 Terms Used to Describe the Mechanical Structure

backplane: A printed-circuit board (pcb) with 96-pin connectors and signal paths that bus (connect corresponding) connector pins. Some systems have a single pcb, J1 backplane. It provides the signal paths needed for basic operation. Other systems also have an optional second pcb, J2 backplane. It provides the additional 96-pin connectors and signal paths needed for wider data and address transfers. Still others have a single pcb, J1/J2 backplane, which provides the signal conductors and connectors of the J1 and J2 backplanes.

board: A printed-circuit board (pcb), its collection of electronic components, and either one or two 96-pin connectors that can be plugged into backplane connectors.

slot: A position where a printed-circuit board (pcb) can be inserted into the backplane. When the system has a J1 and a J2 backplane (or a combination J1/J2 backplane) each slot provides a pair of 96-pin connectors. When the system has only a J1 backplane, then each slot provides a single 96-pin connector.

subrack: A rigid framework that provides mechanical support for printed-circuit boards (pcb) inserted into the backplane, ensuring that the connectors mate properly and that adjacent pcb do not contact each other. It also guides

the cooling airflow through the system and ensures that inserted pcb do not disengage themselves from the backplane due to vibration or shock

1.2.1.2 Terms Used to Describe the Functional Structure

Figure 1 shows a simplified block diagram of the functional structure, including the signal lines, backplane interface logic, and functional modules.

arbiter: A functional module that accepts bus requests from requester modules and grants control of the data transfer bus (DTB) to one requester at a time.

arbitration bus: One of the four buses provided by the backplane. This bus allows an arbiter module and several requester modules to coordinate use of the DTB.

backplane interface logic: Special logic that takes into account the characteristics of the backplane; its signal line impedance, propagation time, termination values, etc. The specification prescribes certain rules for the design of this logic based on the maximum length of the backplane and its maximum number of printed-circuit board (pcb) slots.

bus timer: A functional module that measures the time each data transfer takes on the DTB and terminates the DTB cycle when the transfer time is not within reason. Without this module, when the master attempts to transfer data to or from a nonexistent slave location it could wait forever. The bus timer prevents this delay by terminating the cycle.

daisy-chain: A special type of signal line that is used to propagate a signal level from printed-circuit board (pcb) to pcb, starting with the first slot and ending with the last slot. There are four bus grant daisy-chains and one interrupt-acknowledge daisy-chain on the backplane.

data transfer bus (DTB): One of the four buses provided by the backplane. The data transfer bus allows masters to direct the transfer of binary data among themselves and slaves.

data-transfer-bus cycle: A sequence of level transitions on the signal lines of the DTB that result in the transfer of an address or an address and data between a master and a slave. The data-transfer-bus cycle is divided into two portions (1) the address broadcast and (2) zero or more data transfers.

functional module: A collection of electronic circuitry that resides on one printed-circuit board (pcb) and works together to accomplish a task.

IACK daisy-chain driver: A functional module that activates the interrupt acknowledge daisy-chain whenever an interrupt handler acknowledges an interrupt request. This daisy-chain ensures that only one interrupter responds with its status/ID when more than one has generated an interrupt request.

interrupter: A functional module that generates an interrupt request on the priority interrupt bus, and then provides status/ID information when the interrupt handler requests it.

interrupt handler: A functional module that detects interrupt requests generated by interrupters and responds to those requests by asking for status/ID information.

location monitor: A functional module that monitors data transfers over the data transfer bus (DTB) to detect accesses to the locations it has been assigned to watch. When an access to one of these assigned locations occurs, the location monitor generates an on-board signal.

master: A functional module that initiates data transfer bus (DTB) cycles to transfer data between itself and a slave module.

power monitor: A functional module that monitors the status of the primary power source to the system, and signals when that power has strayed outside the limits required for reliable system operation. Since most systems are powered by an ac source, the power monitor is typically designed to detect drop-out or brown-out conditions on ac lines.

priority interrupt bus: One of the four buses provided by the backplane. The priority interrupt bus allows interrupter modules to send interrupt requests to interrupt-handler modules.

requester: A functional module that resides on the same printed-circuit board (pcb) as an interrupt handler or a master and requests use of the data transfer bus (DTB) whenever its interrupt handler or master needs it.

serial clock driver: A functional module that provides a periodic timing signal that synchronizes the operation of IEEE P1132¹ serial bus. Timing specifications for the SERIAL CLOCK DRIVER of the IEEE P1132 are given in Appendix C. Two backplane signal lines are reserved for use by a serial bus. However, the protocols of the serial bus are completely independent of this standard, and the inclusion of a serial bus is not a required feature of this standard.

slave: A functional module that detects data transfer bus (DTB) cycles initiated by a master and, when those cycles specify its participation, transfers data between itself and the master.

system clock driver: A functional module that provides a 16 MHz timing signal on the utility bus.

system controller board: A board that resides in slot 1 of the backplane and has a system clock driver, an arbiter, an iack daisy-chain driver, and a bus timer. Some also have a serial clock driver, a power monitor, or both.

utility bus: This bus includes signals that provide periodic timing and coordinate the power-up and power-down of the systems. It is one of the four buses provided by the backplane.

1.2.1.3 Types of Cycles

read cycle: A data transfer bus (DTB) cycle that is used to transfer 1, 2, 3, or 4 bytes from a slave to a master. The cycle begins when the master broadcasts an address and an address modifier. Each slave captures the address and the address modifier, and verifies if it will respond to the cycle. If it is intended to respond, it retrieves the data from its internal storage, places it on the data bus and acknowledges the transfer. The master then terminates the cycle.

write cycle: A data transfer bus (DTB) cycle that is used to transfer 1, 2, 3, or 4 bytes from a master to a slave. The cycle begins when the master broadcasts an address and an address modifier and places data on the data transfer bus (DTB). Each slave captures the address and the address modifier and verifies if it will respond to the cycle. If it is intended to respond, it stores the data and then acknowledges the transfer. The master then terminates the cycle.

block read cycle: A data transfer bus (DTB) cycle that is used to transfer a block of bytes ranging in number from 1 to 256 bytes from a to a master. This transfer is executed by using a string of 1-, 2-, or 4-byte data transfers. Once the block transfer is initiated, the master does not release the DTB until all of the bytes have been transferred. This operation differs from a string of read cycles insofar as the master broadcasts only one address and one address modifier (at the beginning of the cycle). The slave then increments this address on each transfer so that the data for the next transfer is retrieved from the next higher location.

block write cycle: A data transfer bus (DTB) cycle used to transfer a block of bytes ranging in number from 1 to 256 bytes from a master to a slave. It uses a string of 1-, 2-, or 4-byte data transfers. Once the block transfer is initiated, the master does not release the DTB until all of the bytes have been transferred. It differs from a string of write cycles insofar as the master broadcasts only one address and one address modifier (at the beginning of the cycle). The slave then increments this address on each transfer so that the data from the next transfer is stored in the next higher location.

read-modify-write cycle: A data transfer bus (DTB) cycle that is used to both read from, and write to, a slave location without permitting any other master to access that location. This cycle is most useful in multiprocessing systems where certain memory locations are used to provide semaphore functions.

address-only cycle: A data transfer bus (DTB) cycle that consists of an address broadcast, but does not have a data transfer. Slaves do not acknowledge address-only cycles and masters terminate the cycle without waiting for an acknowledgment.

interrupt acknowledge cycle: A data transfer bus (DTB) cycle, initiated by an interrupt handler, that reads a status/ID from an interrupter. An interrupt handler generates this cycle whenever it detects an interrupt request from an interrupter and it has control of the DTB.

¹At the time of publication of this standard, IEEE P1132 is in preparation. When IEEE P1132 is approved and published it will become a part of this standard.

1.2.2 Basic Structure

The interface system consists of backplane interface logic, four groups of signal lines referred to as buses, and a collection of functional modules, which can be configured as required. The functional modules communicate with each other using the backplane signal lines.

The functional modules are used as vehicles for discussion of the bus protocol, and need not be considered a constraint to logic design. For example, the designer might choose to design logic that interacts with the bus in the manner described, but uses different on-board signals, or monitors additional backplane signals. Boards can be designed to include various combinations of the defined functional modules.

The functional structure is divided into four categories. Each consists of a bus and its associated functional modules that work together to perform specific duties. Figure 2 shows the functional modules and buses. Each category is briefly summarized in 1.2.1.1 through 1.2.2.

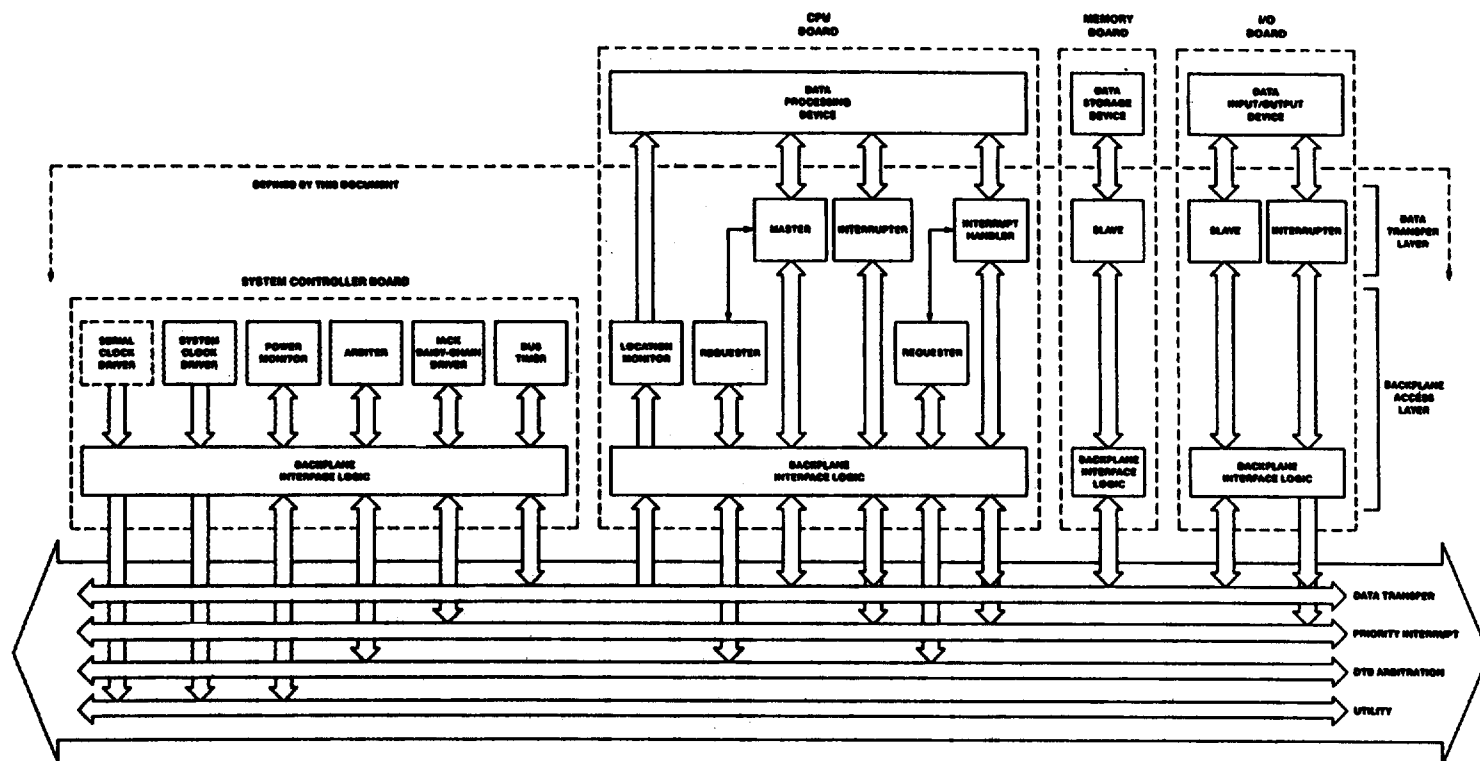


Figure 2—Functional Modules and Buses Defined by the Standard

1.2.2.1 Data Transfer Bus

Devices transfer the data over the data transfer bus D(TB), which contains data and address pathways and associated control signals. The functional modules, masters, slaves, interrupters, and interrupt handlers use the DTB to transfer data among each other. Two other modules, a bus timer and an iack daisy-chain driver, also assist them in this process.

1.2.2.2 DTB Arbitration Bus

Since systems can be configured with more than one master or interrupt handler, a means is provided to transfer control of the DTB among them in an orderly manner, and to guarantee that only one controls the DTB at any given time. The arbitration bus modules, requesters and arbiter, coordinate the control transfer.

1.2.2.3 Priority Interrupt Bus

The priority interrupt capability provides a means by which devices can request service from an interrupt handler. These interrupt requests can be prioritized into a maximum of seven levels. Interrupters and interrupt handlers use the priority-interrupt-bus signal lines.

1.2.2.4 Utility Bus

Periodic clocks, initialization, and failure detection are provided by the utility bus. It includes two clock lines, a system reset line, a system fail line, an ac fail line, and a serial data line.

1.3 Specification Diagrams

Several types of diagrams are used to help define or describe bus operations.

- 1) Timing diagrams show the timing relationships between signal transitions. The times involved have minimum or maximum limits, or both. Some of the times specify the behavior of the backplane interface logic, while others specify the interlocked behavior of the functional modules.
- 2) Sequence diagrams are similar to timing diagrams but only show the interlocked of functional modules. They show a sequence of events, rather than specify the times involved. For example, a sequence diagram indicates that module A cannot generate signal transition B until it detects module C's generation of signal transition D.
- 3) Flow diagrams show a stream of events as they occur during a bus operation. The events are stated in words and result from interaction of two or more functional modules. They describe bus operations in a sequential manner and, at the same time, show how the functional modules interact.

1.4 Terminology

To avoid confusion, and to make very clear the requirements for compliance the following keywords indicate the type of information specified by each category

- 1) Rule
- 2) Recommendation
- 3) Suggestion
- 4) Permission
- 5) Observation

These keywords are used as follows:

Rule. Rules form the basic framework of this standard are sometimes expressed in text form and sometimes in the form of figures, tables, or drawings. All rules shall be followed to ensure compatibility. Rules are characterized by an imperative style. The words *shall* and *shall not* are reserved for stating rules, and are not used for any other purpose.

NOTE — When the rule number is listed the first number corresponds to the section area and the second number uniquely identifies the rule within the section.

Recommendation. Wherever a recommendation is suggested, designers are cautioned to take the advice given. Doing otherwise might result in some awkward problems or poor performance. While the bus has been designed to support high performance systems, it is possible to design a system that complies with all the rules, but has abysmal performance. In many cases, a designer needs a certain level of experience to design printed-circuit boards (pcb) that deliver top performance. Recommendations are based on this type of experience, and are provided to designers to speed their traversal of the learning curve.

NOTE — When the recommendation number is listed the first number corresponds to the section area and the second number uniquely identifies the recommendation within that section.

Suggestion. A suggestion contains advice that is helpful but not vital. The user is encouraged to consider the advice before discarding it. Some design decisions are difficult until experience has been gained. Suggestions are included to help the designer who has not yet gained this experience. Suggestions are concerned with designing printed-circuit boards (pcb) that can be easily reconfigured for operation with other pcb, and with designing the pcb that ease system debugging, etc.

NOTE — When the suggestion number is listed the first number corresponds to the section area and the second number uniquely identifies the suggestion within that section.

Permission. In some cases a rule does not specifically prohibit a certain design approach, but the user might wonder whether that approach will violate the rule, or whether it will lead to some subtle problem. Permissions reassure the user that a certain approach is acceptable, and will not cause problems. The word *may* is reserved for stating permissions, and is not used for any other purpose.

NOTE — When the permission number is listed the first number corresponds to the section area and the second number to the sequence of the permission within that section.

Observation. Observations do not offer any specific advice. They usually follow naturally from what has just been discussed. They spell out the implications of certain rules and bring attention to situations that might otherwise be overlooked. They also give the rationale behind certain rules, so that the user understands why the rule must be followed.

NOTE — When the observation number is listed the first number corresponds to the section area and the second number to the sequence of the observation within that section.

Any text not labeled with one of these keywords describes the system structure or operation. It is written in either a descriptive or a narrative style.

1.4.1 Signal Line State

Bus protocols are described in terms of levels and transitions on bus lines. A signal line is always assumed to be in one of two levels, or in transition between these levels. Whenever the term *high* is used, it refers to a high TTL voltage level. The term *low* refers to a low TTL voltage level. A signal line is *in transition* when its voltage is moving between these levels. (For voltage thresholds see Section 6.).

There are two possible transitions that can appear on a signal line, and these are referred to as *edges*. A rising edge is the time during which a signal makes its transition from a low level to a high level. The falling edge is the time during which a signal makes its transition from a high level to a low level.

Some bus specifications prescribe maximum or minimum rise and fall times for these edges. There is one problem. Printed-circuit board (pcb) designers have very little control over these times. If the backplane is heavily loaded, the rise and fall times will be long. If it is lightly loaded, these times might be short. Even when designers know what the maximum and minimum loading will be, they still need to spend time in the laboratory, experimenting to confirm which drivers provide the needed rise and fall times. Rise and fall times are the result of a complex set of interactions involving the signal line impedances of the backplane, its terminations, the source impedance of the drivers, and the

capacitive loading of the signal line. To trade off all of these factors it is necessary for the pcb designer to study transmission-line theory, and certain specific parameters of drivers and receivers that are not normally found in most manufacturers' data sheets.

Recognizing all of this, this standard does not specify rise and fall times. Instead, it specifies the electrical characteristics for drivers and receivers, and suggests how to design the backplane. It also instructs designers how the worst case bus loading will affect the propagation delay of these drivers so that they can ensure that the required timing is met before building a pcb. When designers follow these guidelines, their pcb will operate reliably with other pcb under worst-case conditions.

1.4.2 Use of the Asterisk

To help define usage, signal mnemonics have an asterisk suffix where required.

- 1) An asterisk following the name of signals that are level significant denotes that the signal is true or valid when the signal is low.
- 2) An asterisk following the name of signals that are edge significant denotes that the actions initiated by that signal occur on a high to low transition.

Observation 1.1 The asterisk is not applicable to the asynchronously running clock lines SYSCLK and SERCLK. There is no fixed phase relationship between these clock lines and other signals.

1.5 Protocol

There are two layers of protocol. The lowest layer, referred to as the backplane access layer, is composed of the backplane interface logic, the utility bus modules, and the arbitration bus modules. The data second layer is the transfer layer, which is composed of the data transfer bus modules and priority interrupt bus modules. Figure 2 shows this layering.

Observation 1.2 The signal lines used by the data transfer layer modules form a special class because they are driven by different modules at different times. They are driven with line drivers that can be turned on and off at each pcb, based upon signals generated in the backplane access layer. It is very important that their turn-on and turn-off times be carefully controlled to prevent two drivers from attempting to drive the same signal line to different levels. Special timing diagram notation is used in this standard to specify their turn-on and turn-off times. It is shown in Fig 3.

There are two basic kinds of protocol on the bus (1) closed-loop protocols and (2) open-loop protocols. Closed-loop protocols use interlocked bus signals while open-loop protocols use broadcast bus signals.

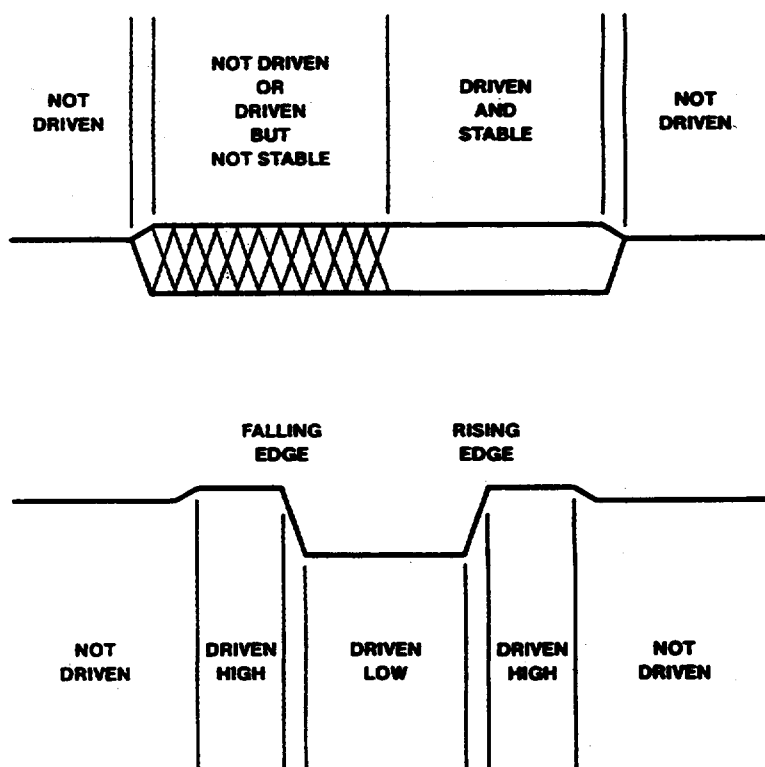


Figure 3—Signal Timing Notation

1.5.1 Interlocked Bus Signal

An interlocked bus signal is sent from a specific module to another specific module. The signal is acknowledged by the receiving module. An interlocked relationship exists between the two modules until the signal is acknowledged. For example, an INTERRUPTER can send an interrupt request that is answered later with an interrupt acknowledge signal (a time limit is not prescribed). The interrupter does not remove the interrupt request until the interrupt handler acknowledges it.

Interlocked bus signals coordinate internal functions of the system, as opposed to interacting with external stimuli. Each interlocked signal has a source module and a destination module within the system.

The address strobe and data strobes are especially important interlocking signals. They are interlocked with the data transfer acknowledge and bus error signals, and coordinate the transfer of addressing information and data that are the basis for all information flow between modules in the data transfer layer.

1.5.2 Broadcast Bus Signal

A module generates a broadcast signal in response to an event. There is no protocol for acknowledging a broadcast signal. Instead, the broadcast is maintained for a minimum specified time that is sufficient to ensure that all appropriate modules detect the signal. Broadcast signals might be activated at any time, irrespective of any other activity taking place on the bus. They are each sent over a dedicated signal line. Some examples are the system reset and ac failure lines. These signal lines are not sent to any specific module, but announce special conditions to all modules.

1.6 System Examples and Explanations

A protocol specification describes, in detail, the behavior of the various functional modules. It discusses the manner in which a module responds to a signal without disclosing the origin of the signal. Because of this procedure, a protocol specification does not give the reader a complete picture. To assist the user, this standard provides examples of typical bus operations. Each example shows one possible sequence of events; other sequences are also possible. However, in providing these examples, there is the danger that users will assume that the sequence shown in the example is the only legal one. To help users avoid this misconception, all examples are given in a narrative style. This is in contrast to the imperative style used when rules for compliance with the standard are given.

2. Data Transfer Bus

2.1 Introduction

This standard includes a high-speed asynchronous parallel data transfer bus (DTB). Figure 4 shows a typical system, including all of the DTB functional module. Masters use the DTB to select storage locations provided by slaves, and to transfer data to or from those locations. Some masters and slaves use all of the DTB lines, while others use only a subset.

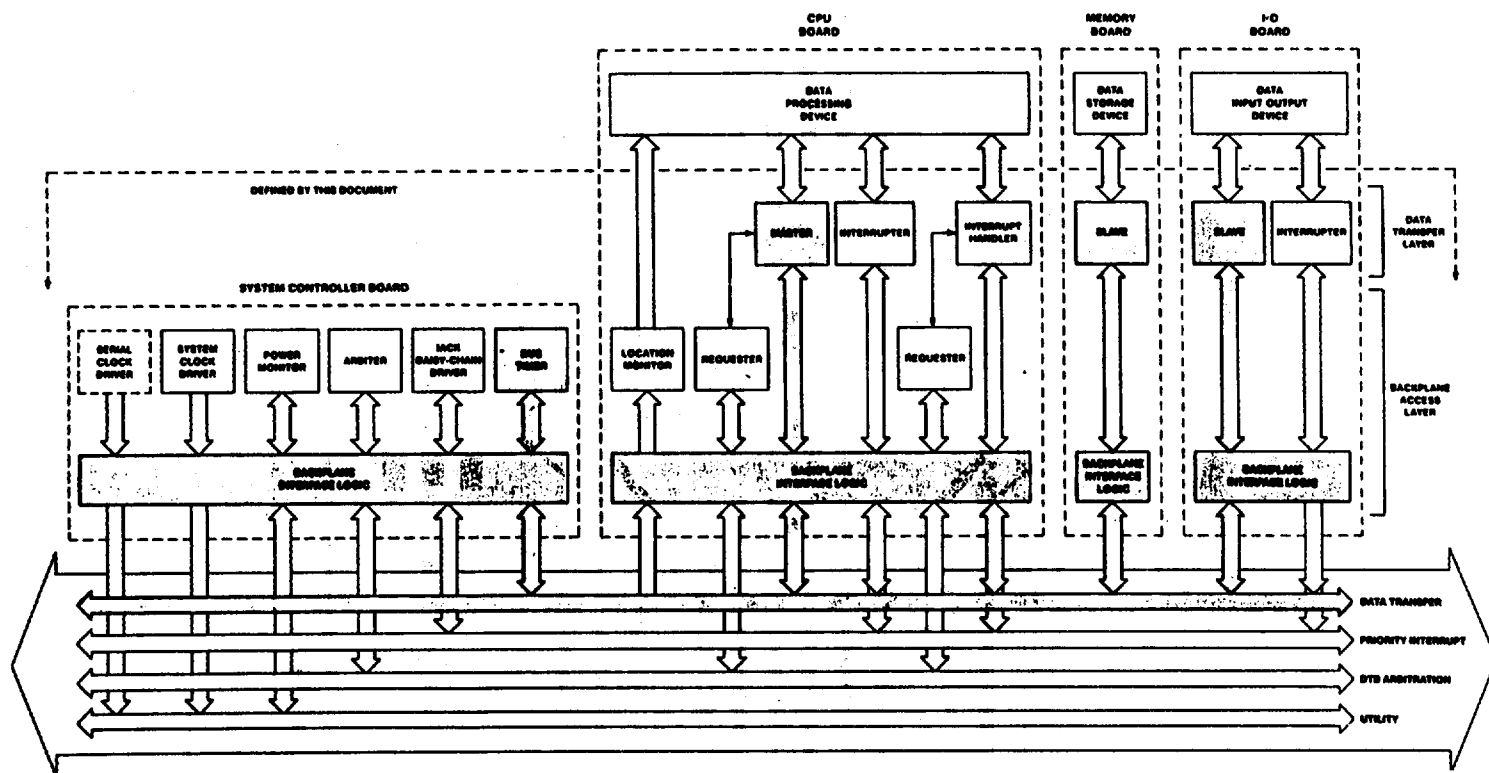


Figure 4—Data Transfer Bus Functional Block Diagram

Location monitors monitor data transfers between masters and slaves. When an access is done to one of the byte locations that it monitors, a location monitor generates an on-board signal. For example, it might signal its on-board processor by means of an interrupt request. In such a configuration, when processor board A writes into a location of the global memory that is monitored by processor B location monitor, processor B will be interrupted.

After a master initiates a data transfer cycle, it waits for the responding slave to respond before finishing the cycle. The asynchronous definition of the bus allows a slave to take all the time it needs to respond. When a slave fails to respond because of some malfunction, or when the master accidentally addresses a location where there is no slave, the bus timer intervenes, allowing the cycle to be terminated.

2.2 Data-Transfer-Bus Lines

The data-transfer-bus lines are grouped into three categories

Addressing Lines	Data Lines	Control Lines
A01–A31	D00–D31	AS*
AM0–AM5		DS0*
DS0*		DS1*
DS1*		BERR*
LWORD*		DTACK*
		WRITE*

Observation 2.1 The two data strobes DS0* and DS1* serve a dual function.

- 1) Their levels are used to select which byte(s) are accessed.
- 2) Their edges are also used as timing signals to coordinate the transfer of data between the master and the slave.

2.2.1 Addressing Lines

The smallest addressable unit of storage is the byte location. Each byte location is assigned a unique binary address. Each byte location can be assigned to one of four categories, according to the least significant two bits of its address (see Table 1).

Table 1—The Four Categories of Byte Locations

Category	Byte Address
BYTE(0)	XXXXXX.....XXXXXX00
BYTE(1)	XXXXXX.....XXXXXX01
BYTE (2)	XXXXXX.....XXXXXX10
BYTE(3)	XXXXXX.....XXXXXX11

A set of byte locations whose addresses differ only in the two least significant bits, is referred to as 4-byte group or a BYTE(–3) group. Some, or all, of the bytes in a 4-byte group can be accessed simultaneously by a single DTB cycle.

Masters use address lines A02–A31 to select which 4-byte group will be accessed. Four additional lines, DS1*, DS0*, A01, and LWORD*, are then used to select which byte location(s) within the 4-byte group are accessed during the data transfer. Using these four lines, a master can access 1-, 2-, 3-, or 4-byte locations simultaneously, as shown in Table 2.

Observation 2.2 When driving both data strobes low, one data strobe might go low slightly after the other. In this case the signal levels indicated in Table 2 are the final levels.

Observation 2.3 Given the 4-signal line levels shown in Table 2, there are 16 possible combinations of levels. Of these 16, there are two illegal combinations that are not used (see Rule 2.1).

Rule 2.1 Masters shall not generate DTB cycles where the final levels of DS0*, DS1*, A01, and LWORD* are either of the following illegal combinations:

DS1*	DS0*	A01	LWORD*
high	low	high	low
low	high	high	low

Permission 2.1 When a master accesses byte locations BYTE(1–2) (see Table 2), it may generate either of the two combinations described in Rule 2.1 briefly as transition states (that is, while one data strobe has fallen, but the other has not).

Observation 2.4 Whenever a master drives LWORD* low and A01 high it drives both data strobes low (Any other combination is illegal). Board designers can take advantage of this to simplify the logic on slaves.

Permission 2.2 To simplify the required logic, slaves that respond to a cycle that accesses byte location BYTE(1–2) (see Table 2) may be designed without logic to distinguish between these cycles and the two illegal cycles described in Rule 2.1.

2.2.2 Address-Modifier Lines

There are six address-modifier lines. They allow the master to pass additional binary information to the slave during DTB cycles. Table 3 lists all of the 64 possible address-modifier (AM) codes and classifies each into one of three categories:

- 1) Defined, including:
 - a) Short addressing AM codes indicate that address lines A02–A15 are being used to select a BYTE(0–3) group
 - b) Standard addressing AM codes indicate that address lines A02–A23 are being used to select a BYTE(0–3) group
 - c) Extended addressing AM codes indicate that address lines A02–A31 are being used to select a BYTE(0–3) group
- 2) Reserved
- 3) User-defined

Table 2—Use of DS0*, DS1*, A01, and LWORD* To Select Byte Locations

Byte Locations Selected	DS1*	DS0*	A01	LWORD*
Single-byte access				
BYTE(0)	low	high	low	high
BYTE(1)	high	low	low	high
BYTE(2)	low	high	high	high
BYTE(3)	high	low	high	high
Double-byte access				
BYTE(0–1)	low	low	low	high
BYTE(1–2)	low	low	high	low
BYTE(2–3)	low	low	high	high
Triple-byte access				
BYTE(0–2)	low	high	low	low
BYTE(1–3)	high	low	low	low
Quad-byte access				
BYTE(0–3)	low	low	low	low

Rule 2.2 Except for the user-defined codes, the codes defined in Table 3 shall not be used for purposes other than those specified.

Rule 2.3 Slaves shall not respond to reserved address-modifier codes.

Observation 2.5 Reserved address-modifier codes are for future enhancements. When slave boards respond to these codes, incompatibilities might result at some future date, when usage of these codes is defined.

Permission 2.3 User-defined codes may be used for any purpose that board vendors or board users deem appropriate (page switching, memory protection, master or task identification, privileged access to resources, etc).

Table 3—Address Modifier Codes

HEX	Address Modifier						Function
Code	5	4	3	2	1	0	
3F	H	H	H	H	H	H	Standard supervisory block transfer
3E	H	H	H	H	H	L	Standard supervisory program access
3D	H	H	H	H	L	H	Standard supervisory data access
3C	H	H	H	H	L	L	Reserved
3B	H	H	H	L	H	H	Standard nonprivileged block transfer
3A	H	H	H	L	H	L	Standard nonprivileged program access
39	H	H	H	L	L	H	Standard nonprivileged data access
38	H	H	H	L	L	L	Reserved
37	H	H	L	H	H	H	Reserved
36	H	H	L	H	H	L	Reserved
35	H	H	L	H	L	H	Reserved
34	H	H	L	H	L	L	Reserved
33	H	H	L	L	H	H	Reserved
32	H	H	L	L	H	L	Reserved
31	H	H	L	L	L	H	Reserved
30	H	H	L	L	L	L	Reserved
2F	H	L	H	H	H	H	Reserved
2E	H	L	H	H	H	L	Reserved
2D	H	L	H	H	L	H	Short supervisory access
2C	H	L	H	H	L	L	Reserved
2B	H	L	H	L	H	H	Reserved
2A	H	L	H	L	H	L	Reserved
29	H	L	H	L	L	H	Short nonprivileged access
28	H	L	H	L	L	L	Reserved
27	H	L	L	H	H	H	Reserved
26	H	L	L	H	H	L	Reserved
25	H	L	L	H	L	H	Reserved
24	H	L	L	H	L	L	Reserved
23	H	L	L	L	H	H	Reserved
22	H	L	L	L	H	L	Reserved

HEX	Address Modifier						Function
Code	5	4	3	2	1	0	
21	H	L	L	L	L	H	Reserved
20	H	L	L	L	L	L	Reserved
1F	L	H	H	H	H	H	User-defined
1E	L	H	H	H	H	L	User-defined
1D	L	H	H	H	L	H	User-defined
1C	L	H	H	H	L	L	User-defined
1B	L	H	H	L	H	H	User-defined
1A	L	H	H	L	H	L	User-defined
19	L	H	H	L	L	H	User-defined
18	L	H	H	L	L	L	User-defined
17	L	H	L	H	H	H	User-defined
16	L	H	L	H	H	L	User-defined
15	L	H	L	H	L	H	User-defined
14	L	H	L	H	L	L	User-defined
13	L	H	L	L	H	H	User-defined
12	L	H	L	L	H	L	User-defined
11	L	H	L	L	L	H	User-defined
10	L	H	L	L	L	L	User-defined
0F	L	L	H	H	H	H	Extended supervisory block transfer
0E	L	L	H	H	H	L	Extended supervisory program access
0D	L	L	H	H	L	H	Extended supervisory data access
0C	L	L	H	H	L	L	Reserved
0B	L	L	H	L	H	H	Extended nonprivileged block transfer
0A	L	L	H	L	H	L	Extended nonprivileged program access
09	L	L	H	L	L	H	Extended nonprivileged data access
08	L	L	H	L	L	L	Reserved

HEX	Address Modifier						Function
Code	5	4	3	2	1	0	
07	L	L	L	H	H	H	Reserved
06	L	L	L	H	H	L	Reserved
05	L	L	L	H	L	H	Reserved
04	L	L	L	H	L	L	Reserved
03	L	L	L	L	H	H	Reserved
02	L	L	L	L	H	L	Reserved
01	L	L	L	L	L	H	Reserved
00	L	L	L	L	L	L	Reserved

L = low-signal level
H = high-signal level

Recommendation 2.1 To allow users to tailor the use of the user-defined address-modifier codes to their own needs and decode them in a flexible way on slave boards. Users can then configure the board to give any decoding required by theft system.

Observation 2.6 Socketed programmable devices provide a flexible way for decoding the address-modifier codes.

Suggestion 2.1 Where slaves are manufactured with a programmed device (for example, a PROM or a FPLA) installed in the socket, program the device so that the slave responds to the following AM codes:

A16 slaves with D08(O), D08(E0), D16, or D32 capability: 29, 2D

A24 slaves with D08(O), D08(E0), D16, or D32 capability: 39, 3A, 3D, and 3E

A32 slaves with D08(O), D08(E0), D16, or D32 capability: 09, 0A, 0D, and 0E

A24 slaves with BLT capability: 3B, 3F

A32 slaves with BLT capability: 0B, 0F

(The mnemonics A16, A24, and A32 are defined in Table 9. The D08(O), D08(E0), D16, D32, and BLT are defined in Tables 10 and 11.

2.2.3 Data Lines

Systems can be built with a backplane configuration that provides either 16 data lines (D00–D15) or 32 data lines (D00–D31). Backplane configurations that provide 16 data lines allow a master to access only 2-byte locations simultaneously, while those with 32 data lines allow it to access up to 4-byte locations simultaneously. When the master has selected 1-, 2-, 3-, or 4-byte locations, using the method described in 2.2.1, it can transfer binary data between itself and those locations over the data bus. Table 4 shows how the data lines are used to access byte locations.

Table 4—Use of the Data Lines to Access Byte Locations

Byte Locations Accessed	D24–D31	D16–D23	D08–D15	D00–D07
BYTE(0)			BYTE(0)	
BYTE(1)				BYTE(1)
BYTE(2)			BYTE(2)	
BYTE(3)				BYTE(3)
BYTE(0–1)			BYTE(0)	BYTE(1)
BYTE(1–2)		BYTE(1)	BYTE(2)	
BYTE(2–3)			BYTE(2)	BYTE(3)
BYTE(0–2)	BYTE(0)	BYTE(1)	BYTE(2)	
BYTE(1–3)		BYTE(1)	BYTE(2)	BYTE(3)
BYTE(0–3)	BYTE(0)	BYTE(1)	BYTE(2)	BYTE(3)

2.2.4 Data-Transfer-Bus Control lines

The following signal lines are used to control the movement of data over the data transfer lines:

AS*	Address Strobe
DS0*	Data Strobe Zero
DS1*	Data Strobe One
BERR*	Bus Error
DTACK*	Data Transfer Acknowledge
WRITE*	Read/Write

2.2.4.1 AS*

A falling edge on AS* informs all slaves that the address is stable and can be captured.

2.2.4.2 DS0* and DS1*

In addition to their function in selecting byte locations for data transfer, as described in 2.2.1, the data strobes also serve additional functions. On write cycles, the first falling edge of a data strobe indicates that the master has placed valid data on the data bus. On read cycles, the first rising edge tells the slave that it can remove its data from the data bus.

Observation 2.7 As specified in 2.6, masters are not permitted to drive either of the data strobes low before driving AS* low. However, due to the fact that AS* might be more heavily loaded on the backplane than the data strobes, slaves and location monitors might detect a falling edge on a data strobe, before they detect the falling edge on AS*.

Permission 2.5 Slaves that do not have block transfer capability (as defined in 2.3.7), and location monitors may be designed to capture the address when they detect a falling edge on a data strobe instead of on the falling edge of AS*.

Observation 2.8 Slaves and location monitors that capture the address on the falling edge of the data strobe(s) need not monitor AS*.

Observation 2.9 To take full advantage of address pipelining as described in 2.4.2, or to perform block read and write cycles, a slave should capture the address on the falling edge of AS*.

2.2.4.3 DTACK*

The slave drives DTACK* low to indicate that it has successfully received the data on a write cycle. On a read cycle, the slave drives it low to indicate that it has placed data on the data bus.

2.2.4.4 BERR*

BERR* is driven low by the slave or by the bus timer to indicate to the master that the data transfer was unsuccessful. For example, when a master tries to write to a location that contains read-only memory, the responding slave can drive BERR* low. When the master tries to access a location that is not provided by any slave, the bus timer drives BERR* low after a specified period.

Suggestion 2.2. Design slaves to respond with a falling edge on BERR* when they detect an uncorrectable error in the data they retrieve from their internal storage during a read cycle.

2.2.4.5 WRITE*

WRITE* is a level-significant signal line that is strobed by the falling edge of the first data strobe. It is used by the master to indicate the direction of data transfer operations. When WRITE* is driven low, the data transfer direction is from the master to the slave. When WRITE* is driven high, the data transfer direction is from the slave to the master.

2.3 DTB Modules — Basic Description

In addition to the address-only cycle, the DTB protocol defines 33 different cycle types that are used to transfer data. Each of these 34 cycle types can be used in any of three addressing modes:

- 1) Short (16 bit)
- 2) Standard (24 bit)
- 3) Extended (32 bit).

The capabilities of masters, slaves, and location monitors are described by a list of mnemonics that show what cycle types they can generate, accept, or monitor respectively. These mnemonics are described in 2.3.5 through 2.3.10.

Block diagrams for the four types of DTB functional modules, that is, master, slave, location monitor, and bus timer, are provided in 2.3.1 and 2.3.4.

Rule 2.7 Output signal lines shown with solid lines in Figs 5, 6, 7, and 8 shall be driven by the module, unless it would always drive them high.

Observation 2.11 When an output signal line is not driven, then terminators on the backplane ensure that it is high.

Rule 2.8 Input signal lines shown with solid lines in Figs 5, 6, 7, and 8 shall be monitored and responded to in the appropriate fashion.

Observation 2.12 Rules and permissions for driving and monitoring signal lines shown with dotted lines in Figs 5, 6, and 8 are given in Tables 5, 6, and 8.

2.3.1 Master

The block diagram of the master is shown in Fig 5. The dotted lines show signals whose use varies among the various types of masters. Table 5 specifies the requirements of the various types of masters to drive and monitor these lines. Further rules describing how the various types of masters drive the address lines are given in Table 20, the data lines in Table 21, and the LWORD*, DS0*, and DS1* and A01 lines in Table 20.

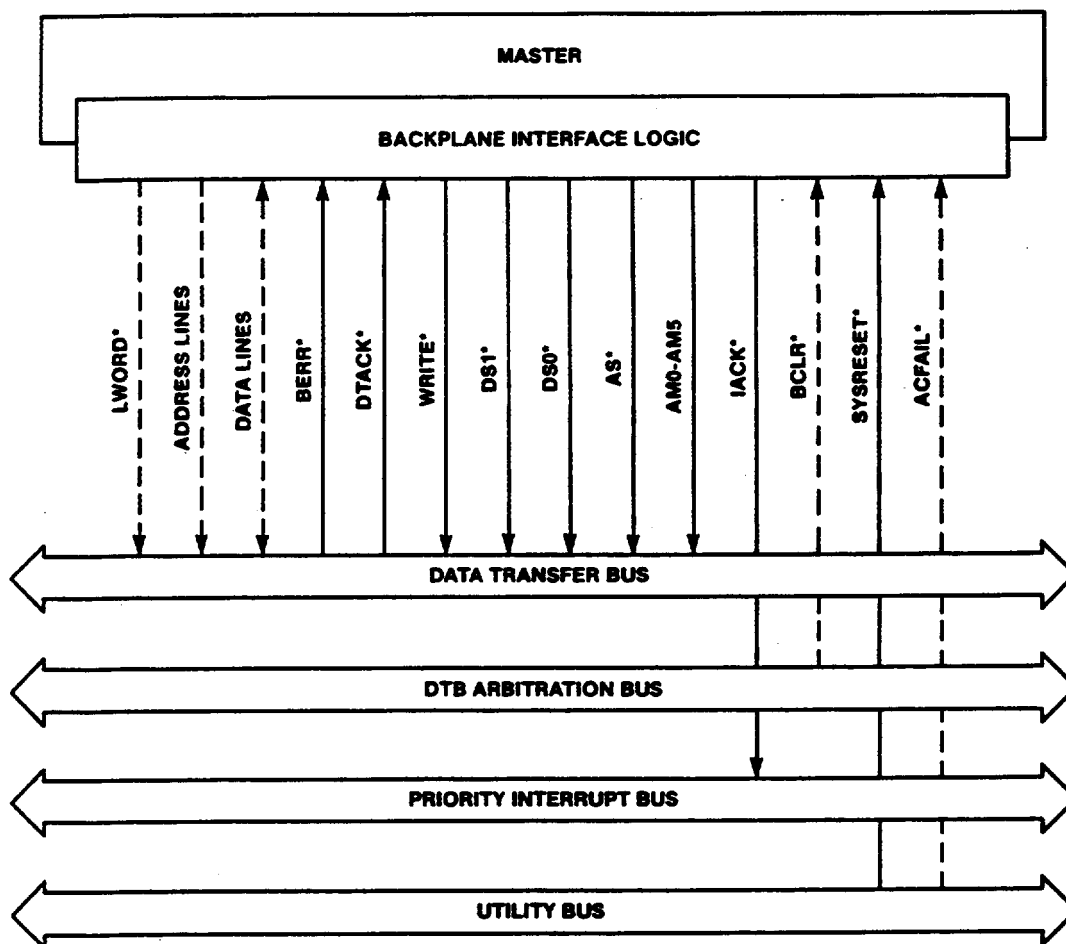


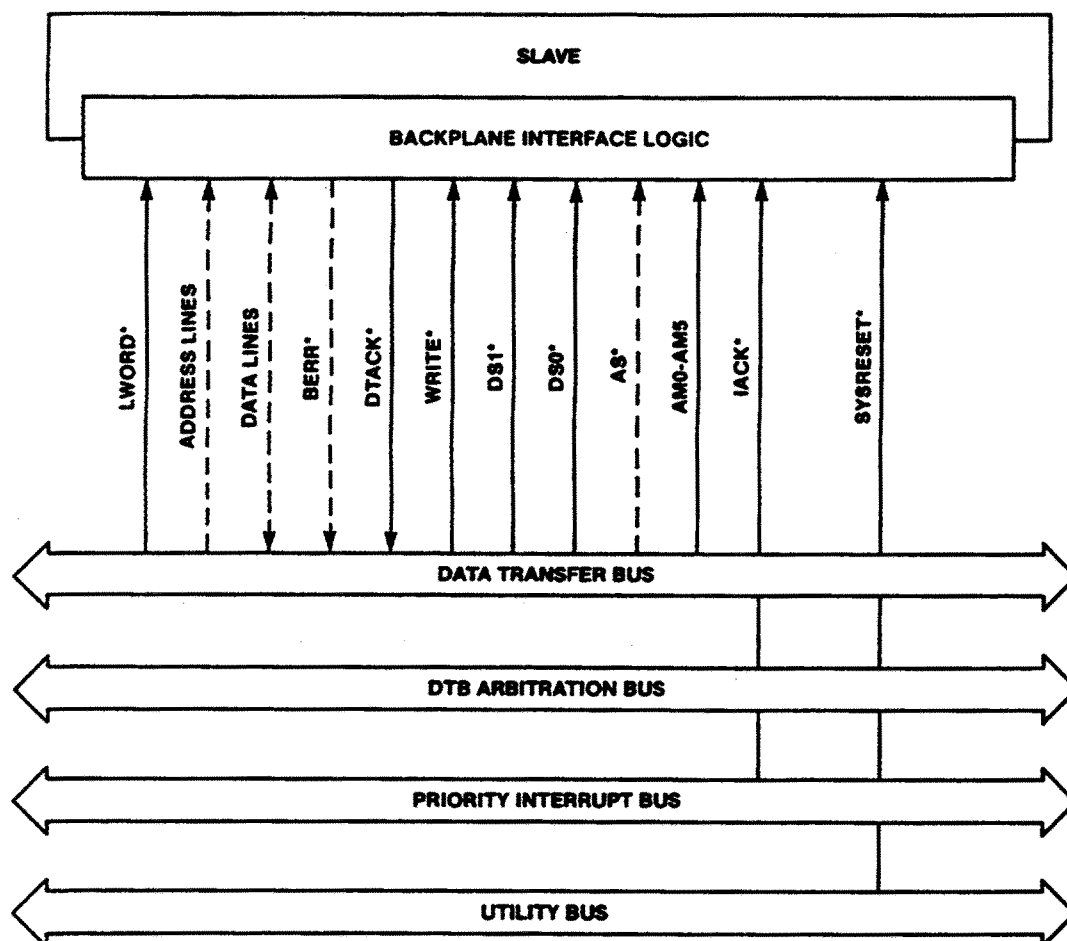
Figure 5—Block Diagram: Master

Table 5—Masters: Rules and Permissions for Monitoring and Driving the Dotted Lines

Type of Master	Rules and Permissions
D08(E0) and D16	Shall monitor and drive D00–D15 May or may not drive LWORD* May or may not drive or monitor D16–D31
D32	Shall drive LWORD* Shall monitor and drive D00–D31
A16	Shall drive A01–A15 May or may not drive A16–A31
A24	Shall drive A01–A23 May or may not drive A24–A31
A32	Shall drive A01–A31
ALL	May or may not monitor BCLR*, or ACFAIL* (see Sections 3. and 5.)

NOTES:

- 1 — The mnemonics D08(E0), D16, and D32 are defined in Table 10.
 2 — The mnemonics A16, A24, and A32 are defined in Table 9.

**Figure 6—Block Diagram: Slave**

2.3.2 Slave

The block diagram of the slave is shown in Fig 6. The dotted lines show signals whose use varies among the various types of slaves. Table 6 specifies the requirements of the various types of slaves to drive and monitor these lines. Further information describing how the various types of slaves drive the data lines is given in Table 21.

Table 6—Slaves; Rules and Permissions for Monitoring and Driving the Dotted Lines

Type of Slave	Rules and Permissions
D08(O)	Shall monitor and drive D00–D07 May or may not monitor or drive D08–D31 May or may not monitor AS*
D08(E0) and D16	Shall monitor and drive D00–D15 May or may not monitor or drive D16–D31 May or may not monitor AS*
D32	Shall monitor and drive D00–D31 May or may not monitor AS*
BLT	Shall monitor AS*
A16	Shall monitor A01–A15 May or may not monitor A16–A31
A24	Shall monitor A01–A23 May or may not monitor A24–A31
A32	Shall monitor A01–A31
ALL	May or may not drive BERR*
NOTES: The mnemonics D08(O), D08(E0), D16, and D32 are defined in Table 10. The mnemonic BLT is defined in Table 11. The mnemonics A16, A24, and A32 are defined in Table 9.	

2.3.3 Bus Timer

The block diagram of the bus timer is shown in Fig 7. Bus timers can be designed to drive BERR* low after various periods of time. Table 7 shows how the mnemonic BTO() is used to describe the various types of bus timers.

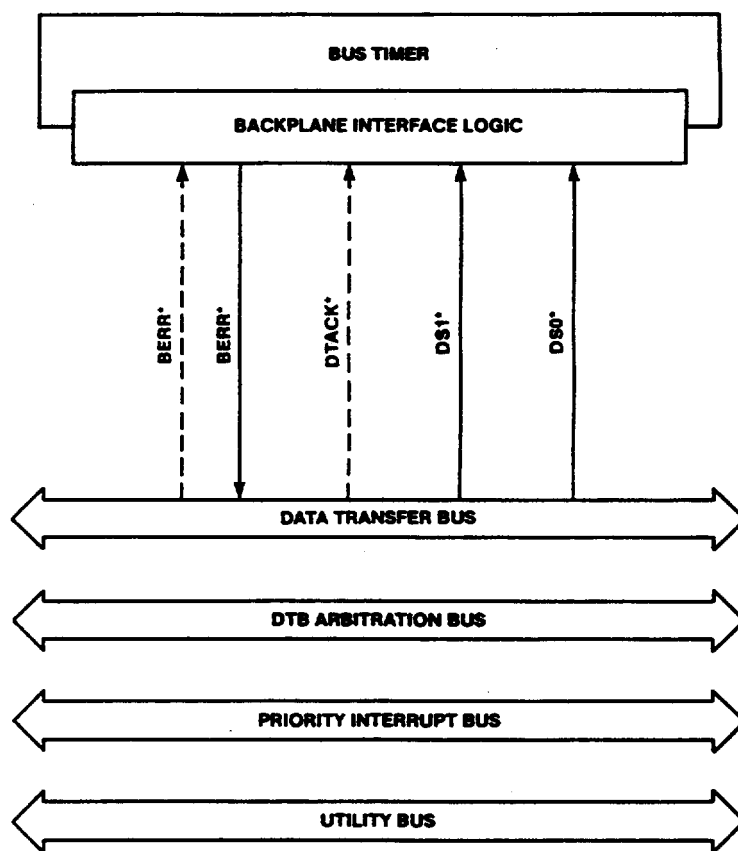


Figure 7—Block Diagram: Bus Timer

Table 7—Use of the BTO() Mnemonic to Specify the Time-Out Period of Bus Timers

The Following Mnemonic	When Applied to a	Means that It
BTO(x)	Bus Timer	Drives BERR* low when the first data strobe stays low for longer than <i>x</i> <i>us</i> .

Observation 2.13 The dotted DTACK* and BERR* lines shown in Fig 7 allow designers to implement a bus timer in one of two ways

- 1) To drive BERR* low when the first data strobe stays low for longer than the bus time-out period, regardless of the levels on the DTACK* and BERR* lines
- 2) To drive BERR* low when the first data strobe stays low for longer than the bus time-out period, but only when both DTACK* and BERR* are high at the point of time-out.

2.3.4 Location Monitor

The block diagram of the location monitor is shown in Fig 8. The dotted lines show signals whose use varies among the various types of location monitors. Table 8 specifies the requirements of the various types of location monitors to monitor these lines.

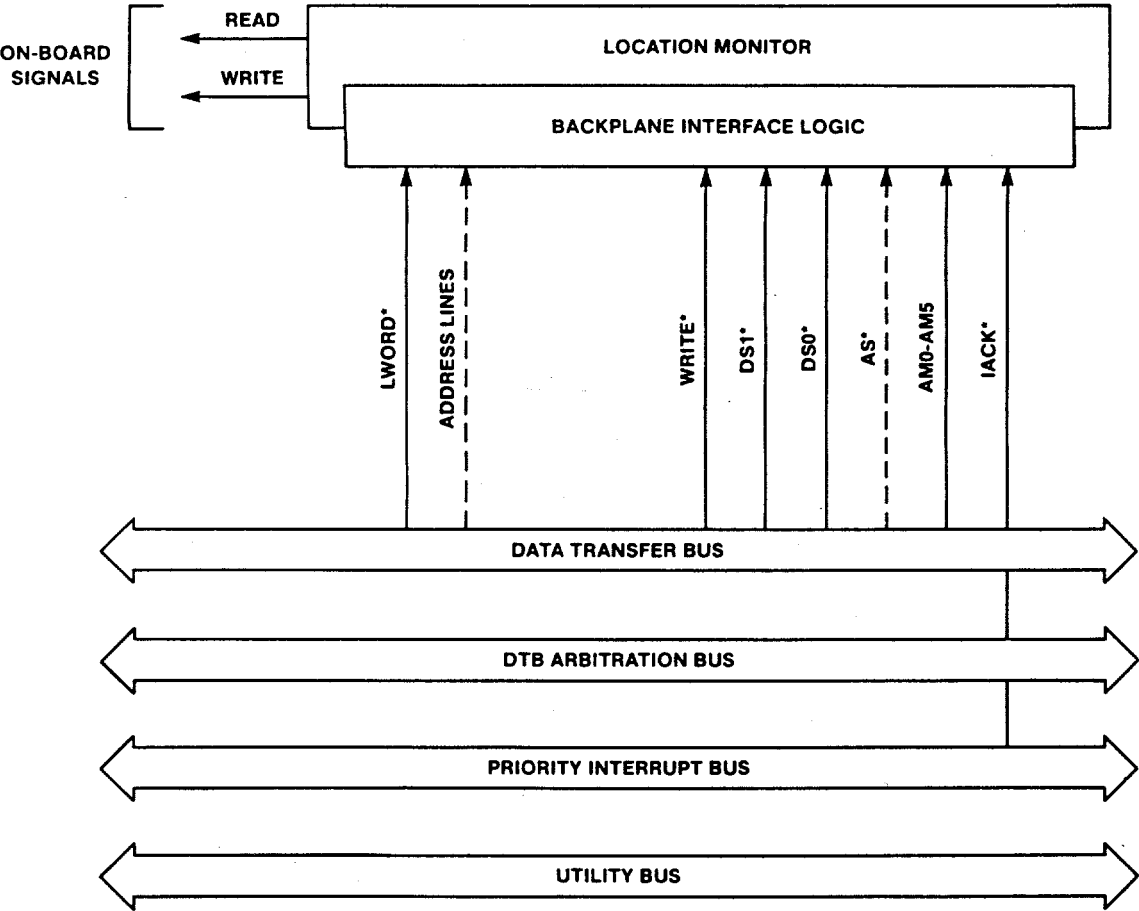


Figure 8—Block Diagram: Location Monitor

Table 8—Location Monitors: Rules and Permissions for Monitoring the Dotted Lines

Type of Location Monitor	Rules and Permissions
A16	Shall monitor A01–A15 May or may not monitor A16–A31
A24	Shall monitor A01–A23 May or may not monitor A24–A31
A32	Shall monitor A01–A31
ALL	May or may not monitor AS*
NOTE — The mnemonics A16, A24, and A32 are defined in Table 9.	

2.3.5 Addressing Modes

Masters broadcast an address over the DTB at the beginning of each cycle. This address might be a 16-bit, a 24-bit, or a 32-bit address, depending upon the capabilities of the master broadcasting it. Sixteen-bit addresses are referred to as short addresses, 24-bit addresses are referred to as standard addresses, and 32-bit addresses are referred to as extended addresses. Table 9 shows the various mnemonics used to describe these capabilities, and how each is used to describe masters, slaves, and location monitors.

Table 9—Mnemonics that Specify Addressing Capabilities

The Following Mnemonics	When Applied to a	Means that It
A16	Master	Can generate cycles with short (6-bit) addresses
	Slave	Can accept cycles with short (16-bit) addresses
	Location Monitor	Can monitor cycles with short (16-bit) addresses
A24	Master	Can generate cycles with standard (24-bit) addresses
	Slave	Can accept cycles with standard (24-bit) addresses
	Location Monitor	Can monitor cycles with standard (24-bit) addresses
A32	Master	Can generate cycles with extended (32-bit) addresses
	Slave	Can accept cycles with extended (32-bit) addresses
	Location Monitor	Can monitor cycles with extended (32-bit) addresses

The master broadcasts an address modifier (AM) code along with each address to inform slaves whether the address is short, standard, or extended.

Short addressing is intended primarily for addressing I/O devices. It allows A16 slaves to be designed with less logic, since they do not have to decode as many address lines. While I/O boards can be designed to decode standard addresses and extended addresses, short addressing usually makes this unnecessary.

Standard and extended addressing modes are intended primarily for addressing memory, although there is no rule against designing I/O boards that respond to these addressing modes. Standard and extended addressing modes allow much larger addressing ranges.

Rule 2.9 Slave boards shall decode all of the address modifier lines.

Observation 2.14 Decoding all the address modifier lines allows a slave to differentiate short addresses, standard addresses, and extended addresses.

Observation 2.15 In addition to the three modes of addressing described here, there is a fourth mode that is used on interrupt acknowledge cycles (see Section 4.). These interrupt acknowledge cycles can be distinguished from data transfer cycles by the fact that the IACK* signal line is low instead of high.

Rule 2.10 Whenever a master broadcasts an address over the address bus, it shall ensure that IACK* is high.

Permission 2.7 The master may either drive IACK* high during the address broadcast or it may leave IACK* undriven (The bus terminators will then hold it high).

Rule 2.11 Slaves shall not respond to DTB cycles when IACK* is low.

Observation 2.86 Many systems include a mixture of A16, A24, and A32 slaves.

Rule 2.61 A32 masters shall include the A24 and A16 capabilities.

Rule 2.62 A24 masters shall include the A16 capability.

Suggestion 2.6 Do not assume that the above rules are known to the readers of product specifications. Rather specify A32 master products as “A32, A24, and A16”. Similarly, specify an A24 master product as “A24 and A16”.

2.3.6 Basic Data Transfer Capabilities

There are four basic data transfer capabilities associated with the DTB: D08(O) (Odd byte only), D08(E0) (Even and Odd byte), D16, and D32. These capabilities allow flexibility when interfacing different types of processors and peripherals to the bus.

Eight-bit processors can be interfaced to the bus as D08(E0) masters. Sixteen-bit processors can be interfaced to the bus as D16 masters. The D16 slave is useful for interfacing 16-bit memory devices or 16-bit I/O devices to the DTB.

Many existing peripheral chips have registers that are only 8 bits wide. While these chips often have several of these registers, they cannot provide the contents of two registers simultaneously when a D16 master attempts to access two adjacent locations with a double-byte read cycle. These 8-bit peripheral IC can be interfaced to the DTB as D08(O) slaves, which provide only BYTE(1) and BYTE(3) locations, and respond only to single-odd-byte accesses (This simplifies the D08(O) slave's interface logic, since single-odd-byte accesses always take place over D00–D07).

Rule 2.63 D32 masters, D32 slaves, and D32 location monitors shall include the D08(E0) and the D16 capabilities.

Rule 2.64 D16 masters, D16 slaves, and D16 location monitors shall include the D08(E0) capability.

Suggestion 2.7 Do not assume that the above rules are known to the readers of product specifications. Rather specify products that have the D32 capability as “D32, D16, and D08(E0)”. Similarly, specify D16 products as “D16 and D08(E0)”.

Rule 2.4 D16 slaves shall not respond by driving DTACK* low during cycles that request access to byte locations BYTE(1–2), BYTE(0–2), BYTE(1–3), or BYTE(0–3).

Rule 2.5 D08(E0) slaves shall not respond by driving DTACK* low during cycles that request access to byte locations BYTE(0–1), BYTE(1–2), BYTE(2–3), BYTE(0–2), BYTE(1–3), or BYTE(0–3).

Rule 2.65 D08(O) slaves shall not respond with a falling edge on DTACK* during cycles that request access to byte locations BYTE(O), BYTE(2), BYTE(0–1), BYTE(1–2), BYTE(2–3), BYTE(0–2), BYTE(1–3), or BYTE(0–3).

Suggestion 2.8 Design slaves to respond with a falling edge on BERR* in the following situations:

- 1) When a D08(O), D08(E0), or D16 slave is requested to do a quad-byte cycle
- 2) When a D08(O) or D08(E0) slave is requested to do a double-byte cycle
- 3) When a D08(O), D08(E0), or D16 slave is requested to do an unaligned transfer (That is, a triple-byte transfer or a double-byte BYTE(1–2) transfer)

Table 10 shows the various mnemonics used to describe the basic-data-transfer capabilities, and the function of each to describe masters, slaves, and location monitors.

Table 10—Mnemonics that Specify Basic-Data-Transfer Capabilities

The Following Mnemonic	When Applied to a	Means that It
D08(O)	Slave	Can accept the following cycles: Single-byte read cycles: BYTE(1) READ BYTE(3) READ Single-byte write cycles: BYTE(1) WRITE BYTE(3) WRITE
D08(E0)	Master Slave Location Monitor	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles: Single-byte read cycles: BYTE(0) READ BYTE(1) READ BYTE(2) READ BYTE(3) READ Single-byte write cycles: BYTE(0) WRITE BYTE(1) WRITE BYTE(2) WRITE BYTE(3) WRITE
D16	Master Slave Location Monitor	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles: Double-byte read cycles: BYTE(0–1) READ BYTE(2–3) READ Double-byte write cycles: BYTE(0–1) WRITE BYTE(2–3) WRITE
D32	Master Slave Location Monitor	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles: Quad-byte read cycle: BYTE(0–3) READ Quad-byte write cycle: BYTE(0–3) WRITE

Observation 2.16 It might seem logical to define *even-byte-only* slaves that respond to the even-byte memory locations adjacent to the D08(O) slaves. But, this cannot be done because there is only one DTACK* line. When a master selects both an even-byte and an odd-byte location simultaneously, by doing a double-byte transfer, both slaves drive DTACK* at the same time, and the master doesn't know whether both boards have acknowledged the access.

Observation 2.17 Since D08(O) slaves respond only to odd-byte addresses, they cannot provide contiguous memory. The D08(O) slaves are useful only for I/O, status, or control registers, while D08(E0), D16, and D32 slaves are also useful for memory.

2.3.7 Block-Transfer Capabilities

Masters often access several memory locations in ascending order. When this is the case, block transfer cycles are very useful. They allow the master to provide a single address, and then access data in that location and those at higher addresses, without providing additional addresses.

When a master initiates a block transfer cycle, the responding slave latches the address into an on-board address counter. The master, upon completing the first data transfer, (that is, driving the data strobes high) does not allow the address strobe to go high. Instead, it repeatedly drives the data strobe(s) low in response to data transfer acknowledgments from the slave, and transfers data to or from sequential memory locations in ascending order. To access the next location(s), the slave increments an on-board counter that generates the address for each transition of the data strobe(s).

Observation 2.18 Block transfer cycles of indefinite length complicate the design of memory boards. Specifically, all block transfer slaves (the one that responds and those that do not) need to latch the initial address and then increment the address counter on each bus transfer. All slaves then have to decode the incremented address to determine whether the transfer has crossed a board boundary into their address range. While this is certainly possible, such address decoding typically limits access times of the slave. To simplify the design of these slaves and to permit faster access times, Rule 2.12 was formulated.

Rule 2.12 Block transfer cycles shall not cross any 256-byte boundary

Observation 2.19 Rule 2.12 limits the maximum length of block transfers to 256 bytes. However, knowing that only A01 through A07 will change during the course of the block transfer simplifies the design of block transfer slaves. The upper address lines only have to be decoded once, at the beginning of the block transfer cycle, allowing faster access times on all subsequent data transfers.

Observation 2.20 In some cases it might be necessary to transfer a large block of data that crosses one or more 256-byte boundaries. In such a case, when the hardware on the board that does the block transfer is designed to recognize the arrival at 256-byte boundary, it can momentarily drive AS* high and then initiate another block transfer without the intervention of system software.

The block read cycle is very similar to a string of read cycles. Likewise, the block write cycle is very similar to a string of write cycles. The difference is that only the initial address is broadcast by the master and the address strobe is held low during all of the data transfers.

Observation 2.21 Control of the DTB cannot be transferred during block transfer cycles because AS* is held low through all of the data transfers, and control of the DTB can only be transferred while AS* is high.

Rule 2.66 Slaves that include block transfer capability shall monitor AS*, and shall capture the addressing information when they detect a falling edge on AS*.

Observation 2.86 During single-byte block transfers, data is transferred 8 bits at a time over D00–D07 or D08–D15. One example is as follows:

	D08–D15	D00–D07
First data transfer		BYTE (1)
	BYTE(2)	
		BYTE(3)
	BYTE(0)	
		BYTE(1)
	BYTE(2)	
Last data transfer		BYTE(3)

Table 11 – Mnemonics That Specify Basic-Data-Transfer Capabilities

The Following Mnemonic	When Applied to a	Means that It
BLT	D08(E0) MASTER D08(E0) SLAVE D08(E0) LOC MON	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles:
		Block read cycles: SINGLE-BYTE BLOCK READ Block write cycles: SINGLE-BYTE BLOCK WRITE
	D16 MASTER D16 SLAVE D16 LOC MON	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles:
		Block read cycles: DOUBLE-BYTE BLOCK READ Block write cycles: DOUBLE-BYTE BLOCK WRITE
	D32 MASTER D32 SLAVE D32 LOC MON	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles:
		Block read cycles: QUAD-BYTE BLOCK READ Block write cycles: QUAD-BYTE BLOCK WRITE

Observation 2.87 During double-byte block transfers, data is transferred 16 bits at a time over D00–D15. One example is as follows:

	D08–D15	D00–D07
First data transfer	BYTE(2)	BYTE(3)
	BYTE(0)	BYTE(1)
	BYTE(2)	BYTE(3)
	BYTE(0)	BYTE(1)
	BYTE(2)	BYTE(3)
Last data transfer	BYTE(0)	BYTE(1)

Table 11 lists the mnemonic used to describe block transfer capabilities and how it is used to describe the various types of masters, slaves, and location monitors.

2.3.8 Read-Modify-Write Capability

In multiprocessor systems that share resources, such as memory and I/O, a method is needed to allocate these resources. One very important goal of this allocation algorithm is to ensure that a resource being used by one task cannot be used by another at the same time. The problem is best described by the following example:

Two processors in a multiprocessing system share a common resource (for example, a printer). Only one processor can use the resource at a time. The resource is allocated by a byte in memory, that is, when the byte is set, the resource is busy; when it is cleared, the resource is available. To gain use of the resource, processor A reads the byte and tests it to determine whether it is cleared. When the byte is cleared, processor A sets the byte to lock out processor B. This operation takes two data transfers; a read to test the byte, and a write to set the byte. However, difficulty might arise when the bus is given to processor B between these two transfers. Processor B might then also find the byte cleared and assume the resource is available. Both processors will then set the byte in the next available cycle and attempt to use the resource.

Table 12—Mnemonic That Specifies Read-Modify-Write Capabilities

The Following Mnemonic	When Applied to a	Means that It
RMW	D08(O) SLAVE	Can accept the following cycles: Single-odd-byte RMW cycles: BYTE(1) READ-MODIFY-WRITE BYTE(3) READ-MODIFY-WRITE
	D08(E0) MASTER D08(E0) SLAVE D08(E0) LOC MON	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles: Single-byte RMW cycles: BYTE(0) READ-MODIFY-WRITE BYTE(1) READ-MODIFY-WRITE BYTE(2) READ-MODIFY-WRITE BYTE(3) READ-MODIFY-WRITE
	D16 MASTER D16 SLAVE D16 LOC MON	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles: Double-byte RMW cycles: BYTE(0–1) READ-MODIFY-WRITE BYTE(2–3) READ-MODIFY-WRITE
	D32 MASTER D32 SLAVE D32 LOC MON	Can generate the following cycles: Can accept the following cycles: Can monitor the following cycles: Quad-byte RMW cycles: BYTE(0–3) READ-MODIFY-WRITE

This conflict is avoided by defining a read-modify-write cycle that prevents transferring control of the DTB between the read portion and the write portion of the cycle. This cycle is very similar to a read cycle immediately followed by a write cycle. The difference is that the address strobe is held low during both transfers. This ensures that, unlike a read cycle followed by a write cycle, control of the DTB cannot be transferred during a read-modify-write cycle, as this is only possible while the address strobe is high.

Table 12 lists the mnemonic used to describe read-modify-write capabilities and how it is used to describe the various types of masters, slaves, and location monitors.

2.3.9 Unaligned Transfer Capability

Some 32-bit microprocessors store and retrieve data in an unaligned fashion. For example, a 32-bit value might be stored in four different ways, as shown in Fig 9.

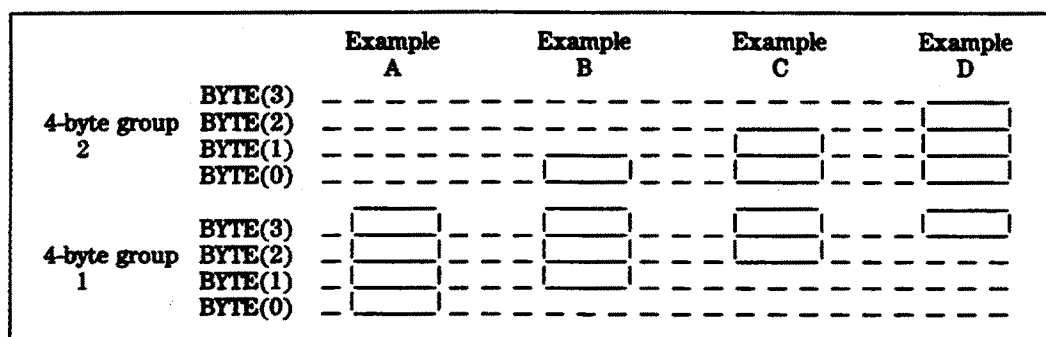


Figure 9—Four Ways That 32 Bits of Data Might Be Stored In Memory

The master can transfer the 32 bits of data using several different sequences of DTB cycles. For example, it can transfer the data one byte at a time, using four single-byte data transfers. However, a master can accomplish the transfer much quicker by using one of the cycle sequences shown in Table 13.

Observation 2.22 The sequences shown in Table 13 is typical of a master that accesses the byte locations in ascending order. The DTB protocols do not require this.

As shown in Table 13, each of these 32-bit transfers can be accomplished with a combination of single-byte and double-byte transfers. However, examples B and D require three bus cycles when this procedure is followed. Because of this, the DTB protocol also includes two triple-byte transfer cycles. When used in combination with a single-byte cycle, these triple-byte cycles allow data to be stored as shown in examples B and D using only two bus cycles.

Some 32-bit microprocessors also store and retrieve data 16 bits at a time, in an unaligned fashion, as shown in Fig 10.

Table 13—Transferring 32 Bits of Data Using Multiple-Byte Transfer Cycles

Example	Cycle Sequences Used to Accomplish the Transfer	Data Bus Lines Used	Byte Locations Accessed (See Fig 9)
A	Quad-byte transfer	D00–D31	Grp 1, BYTE(0–3)
B	Single-byte transfer	D00–D07	Grp 1, BYTE(1)
	Double-byte transfer	D00–D15	Grp 1, BYTE(2–3)
	Single-byte transfer	D08–D15	Grp 2, BYTE(0)
	or		
	Triple-byte transfer	D00–D23	Grp 1, BYTE(1–3)
C	Double-byte transfer	D00–D15	Grp 1, BYTE(2–3)
	Double-byte transfer	D00–D15	Grp 2, BYTE(0–1)
D	Single-byte transfer	D00–D07	Grp 1, BYTE(3)
	Double-byte transfer	D00–D15	Grp 2, BYTE(0–1)
	Single-byte transfer	D08–D15	Grp 2, BYTE(2)
	or		
	Single-byte transfer	D00–D07	Grp 1, BYTE(3)
	Triple-byte transfer	D08–D31	Grp 2, BYTE(0–2)

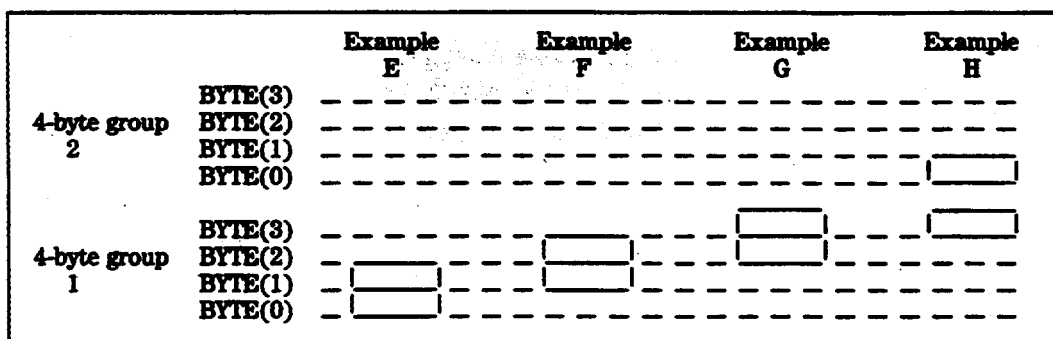


Figure 10—Four Ways That 16 Bits of Data Might Be Stored in Memory

The master can transfer the 16 bits of data using several different sequences of DTB cycles as listed in Table 14.

Observation 2.23 The sequences listed in Table 14 is typical of a master that accesses the byte locations in ascending order. The DTB protocols do not require this.

As shown in Table 14, the 16-bit transfer in example F can be accomplished with two single-byte transfers. However, this requires two bus cycles. Because of this, the DTB protocol also includes a double-byte transfer cycle that allows data to be stored as shown in example F using only one bus cycle.

Table 14—Transferring 16 Bits of Data Using Multiple-Byte Transfer Cycles

Example	Cycle Sequences Used to Accomplish the Transfer	Data Bus Lines Used	Byte Locations Accessed (See Fig 10)
E	Double-byte transfer	D00–D15	Grp 1, BYTE(0–1)
F	Single-byte transfer Single-byte transfer or Double-byte transfer	D00–D07 D08–D15 D08–D23	Grp 1, BYTE(1) Grp 1, BYTE(2) Grp 1, BYTE(1–2)
G	Double-byte transfer	D00–D15	Grp 1, BYTE(2–3)
H	Single-byte transfer Single-byte transfer	D00–D07 D08–D15	Grp 1, BYTE(3) Grp 2, BYTE(0)

Observation 2.24 Since unaligned transfers make use of all 32 data lines, only D32 masters and D32 slaves can do unaligned transfers.

Rule 2.67 D32 slaves and location monitors shall include UAT capability.

Rule 2.6 D08(O), D08(E0), and D16 slaves shall not respond by driving DTACK* low during a cycle that accesses byte locations BYTE(1–2), BYTE(0–2), or BYTE(1–3).

Table 15 lists how the unaligned transfer (UAT) mnemonic is used to describe masters.

Table 15—Mnemonic That Specifies Unaligned Transfer Capability

The Following Mnemonic	When Applied to a	Means that It
UAT	D32 MASTER	Can generate the following cycles: Triple-byte read cycles: BYTE(0–2) READ BYTE(1–3) READ Triple-byte write cycles: BYTE(0–2) WRITE BYTE(1–3) WRITE Double-byte read cycle: BYTE(1–2) READ Double-byte write cycle: BYTE(1–2) WRITE

2.3.10 Address-Only Capability

The address-only cycle is the only cycle on the DTB that is not used to transfer data. It begins as a typical DTB cycle, with the address, address modifier code, IACK* and LWORD* lines becoming valid and AS* falling after a setup time. However, the data strobes are never driven low. After holding the various lines strobed by AS* stable for a prescribed minimum period, the master terminates the cycle without waiting for DTACK* or BERR* to go low (The address-only cycle is also the only type of DTB cycle that does not require a response to complete). Table 16 shows how the mnemonic ADO is used to describe masters.

Table 16—Mnemonic That Specifies Address Only Capability

The Following Mnemonic	When Applied to a	Means that It
ADO	MASTER	Can generate address-only cycles.

Observation 2.25 Address-only cycles can be used to enhance board performance by allowing a CPU board that is already in control of the bus to broadcast an address before it has determined whether or not that address selects a slave on the bus. Broadcasting the address in this fashion allows slaves to decode the address concurrently with the CPU board.

Rule 2.68 All slaves shall be designed to accommodate ADO cycles without loss of data or erroneous separation.

2.3.11 Interaction Between DTB Functional Modules

Data transfers take place between masters and slaves. The master is the module controlling the transfer. The slave that recognizes the address as its own is the responding slave, and all other slaves are nonresponding slaves.

After initiating a data transfer cycle, the master waits for a response from the responding slave. When the master detects that response, it drives its data strobes and address strobe high, terminating the cycle. The slave responds by releasing its response line.

Observation 2.26 Although the timing for the address and the data are largely independent, there are two exceptions:

- 1) The master waits until it has driven AS* low before driving either DS0* or DS1* low
- 2) The slave acknowledges both the AS*, and DS0* and DS1* with either DTACK* or BERR*.

Rule 2.13 If a slave responds to a data transfer cycle, then it shall either drive DTACK* low or it shall drive BERR* low, but it shall not drive both low.

Observation 2.27 Because of possible bus skew due to different loading of AS* and of DS0* and DS1*, the falling edge of DS0*, DS1*, or both, might be detected by the slave slightly before the falling edge on AS*.

Observation 2.28 The WRITE* line is high to identify a read cycle, and low to identify a write cycle before the first data strobe is driven low, and remains stable until both data strobes are high.

Rule 2.14 Before driving the data bus, the master shall ensure that the previous responding slave has stopped driving the data bus. It does so by verifying that DTACK* and BERR* are both high before it drives either the DS0* or DS1*, or both, low on any cycle, and before it drives any of D00–D31 during a write cycle.

Rule 2.15 At the end of a read cycle the responding slave shall release the data bus before allowing DTACK* to go high.

Rule 2.16 During read cycles, the responding slave shall maintain valid data on the data bus until it detects the first data strobe high.

Suggestion 2.3 For optimum performance, design masters so that they drive DS0* and DS1* high as soon as possible after DTACK* or BERR* goes low. Also, design slaves so that they release the data bus and DTACK* as soon as possible after detecting that DS0* and DS1* are high. This allows the maximum data transfer rate on the bus.

Observation 2.29 Addressing information on the bus might change soon after a module drives DTACK* or BERR* low, and before the master drives DS0* and DS1* or AS* high.

A third type of module, the location monitor, monitors the data transfer and generates either, or both, of two on-board signals whenever a byte location that it monitors is accessed. When the access is a write cycle, then the on-board WRITE signal is generated. When the access is a read cycle, then the on-board READ signal is generated. When a read-modify-write cycle is performed, then both on-board signals are generated.

When the cycle takes too long, a fourth module, a bus timer, intervenes by driving BERR* low, completing the data-transfer handshake, and allowing the bus to resume operation.

Rule 2.17 There is a strict interlock between the rising and falling edges of the DS0*/DS1* and DTACK*/BERR*. Before driving either DS0* or DS1* low, a master shall ensure that both DTACK* and BERR* are high. Once it has driven either DS0* or DS1* low, it shall not drive them high and finish a transfer without first detecting DTACK* or BERR* low.

Observation 2.30 A board containing a processor, which directs data transfers between itself and other boards, contains a master module. When the same board also contains memory accessible from the bus, it also contains a slave module. An intelligent peripheral controller might receive commands through a slave interface from a general-purpose processor board. It then might act as a master to access global memory to execute the command it has been given.

2.4 Typical Operation

Masters initiate data transfers over the DTB. The addressed slave then acknowledges the transfer. After receiving the data-transfer acknowledge, the master terminates the data transfer cycle. The asynchronous nature of the DTB allows the slave to control the time taken for the transfer.

Before initiating any data transfers, a master has to be granted exclusive control of the DTB. This ensures that multiple masters will not try to use the DTB at the same time. The master gains control of the DTB using the modules and signal lines of the arbitration bus (This is explained in more detail in 2.5). The following discussion presumes that the master has already been granted and assumed control of the DTB.

2.4.1 Typical Data-Transfer Cycles

Figure 11 shows a typical single-byte read cycle. To start the transfer, the master drives the addressing lines with the desired address and address modifier code. Since this example is a BYTE(1) read cycle, the master drives LWORD* high and A01 low. Since it is not doing an interrupt acknowledge cycle, it does not drive IACK* low. The master then waits for a specified setup time before driving AS* low, to allow the address lines and the address modifier lines to stabilize before the slaves sample them.

Each slave determines whether it should respond by examining the levels on the address lines, the address modifier lines, and IACK*. While this is happening, the master drives WRITE* high to indicate a read operation. The master then verifies that DTACK* and BERR* are high to ensure that the slave from the previous cycle is no longer driving the data bus. When this is the case, the master then drives DS0* low, while keeping DS1* high.

The responding slave then determines which 4-byte group and which byte of that group is to be accessed, and starts the transfer. After it has retrieved the data from its internal storage and placed it on data lines D00–D07, the slave signals the master by driving DTACK* low. The slave then holds DTACK* low and maintains the data valid for as long as the master holds DS0* low.

When the master receives DTACK* driven low, it captures the data on D00–D07, releases the address lines, and drives DS0* and AS* high. The slave responds by releasing D00–D07 and releasing DTACK* high.

Observation 2.31 The master in Fig 11 releases all of the DTB lines at the end of the data transfer. This is not required unless the master's requester released BBSY* during the data transfer as described in 2.5.

The flow for double-byte and quad-byte data transfer cycles are very similar to the single-byte cycle. Flow diagrams for these cycles are shown in Figs 12 and 13.

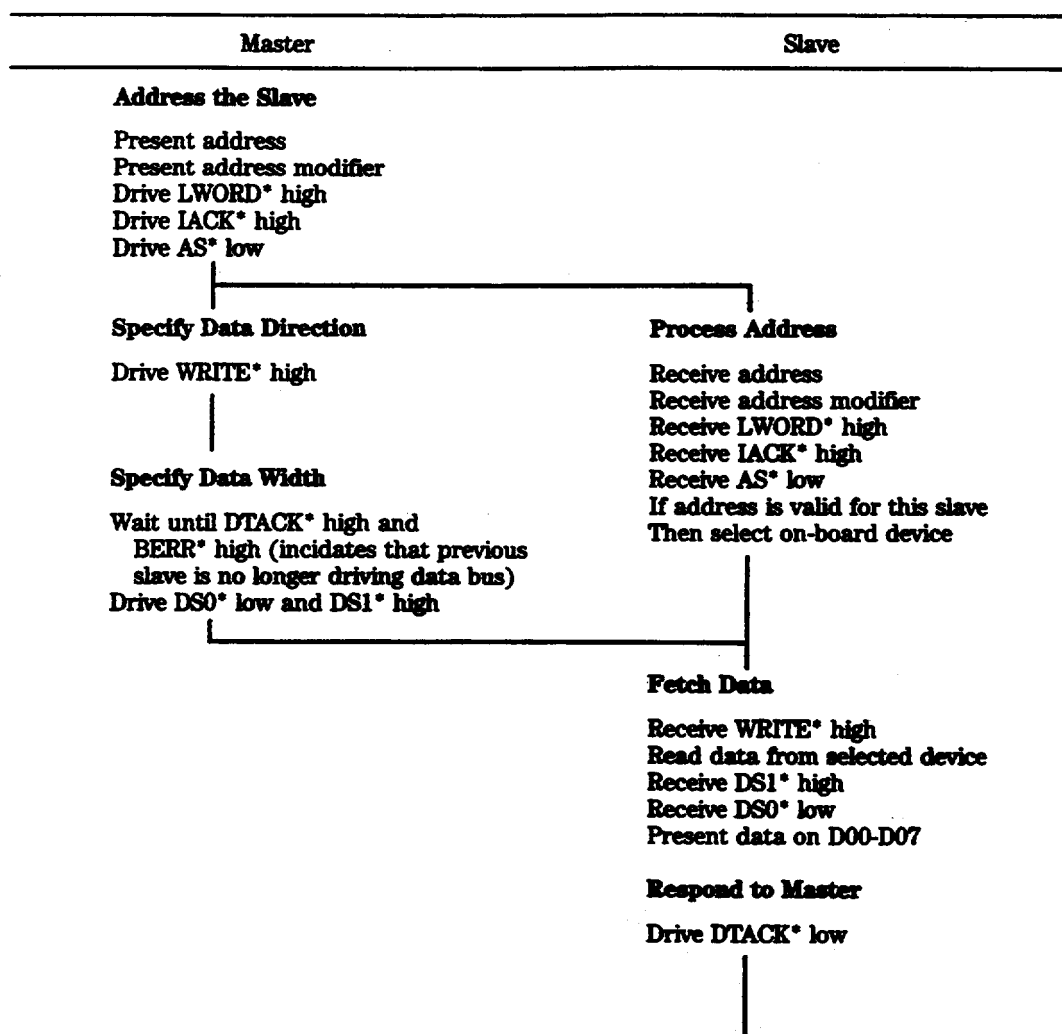


Figure 11—An Example of a Single-Byte Read Cycle

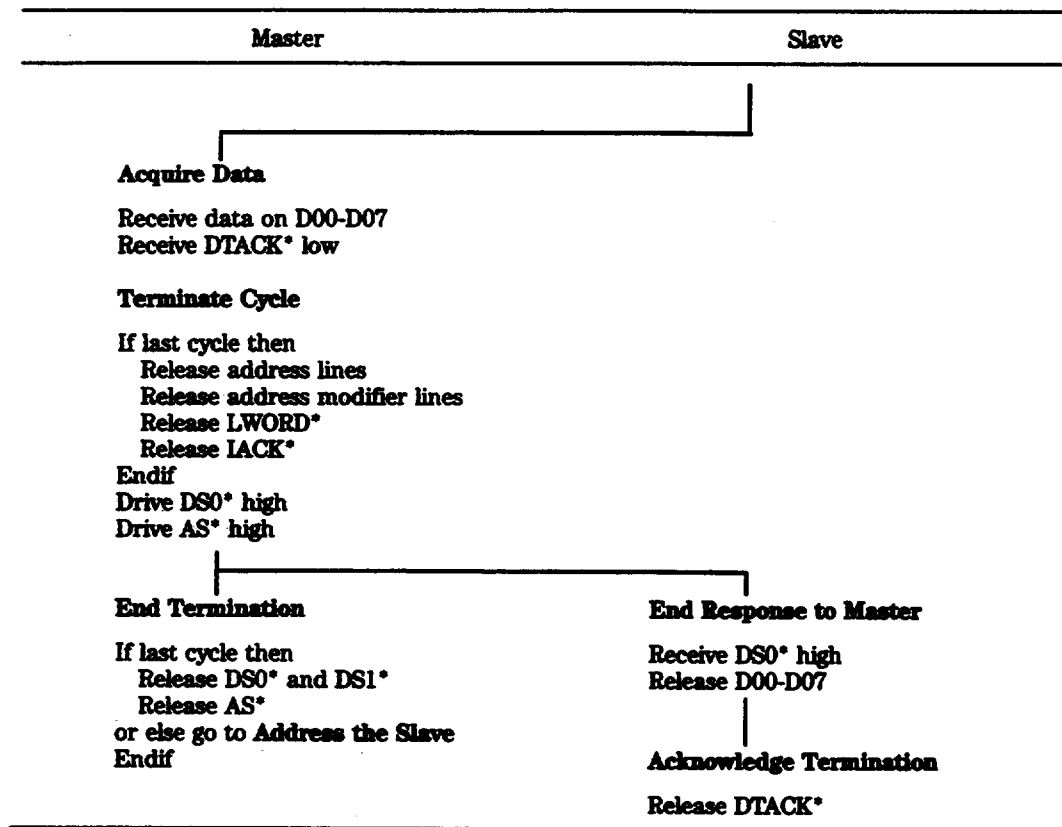


Fig 11 (Continued)—An Example of a Single-Byte Read Cycle

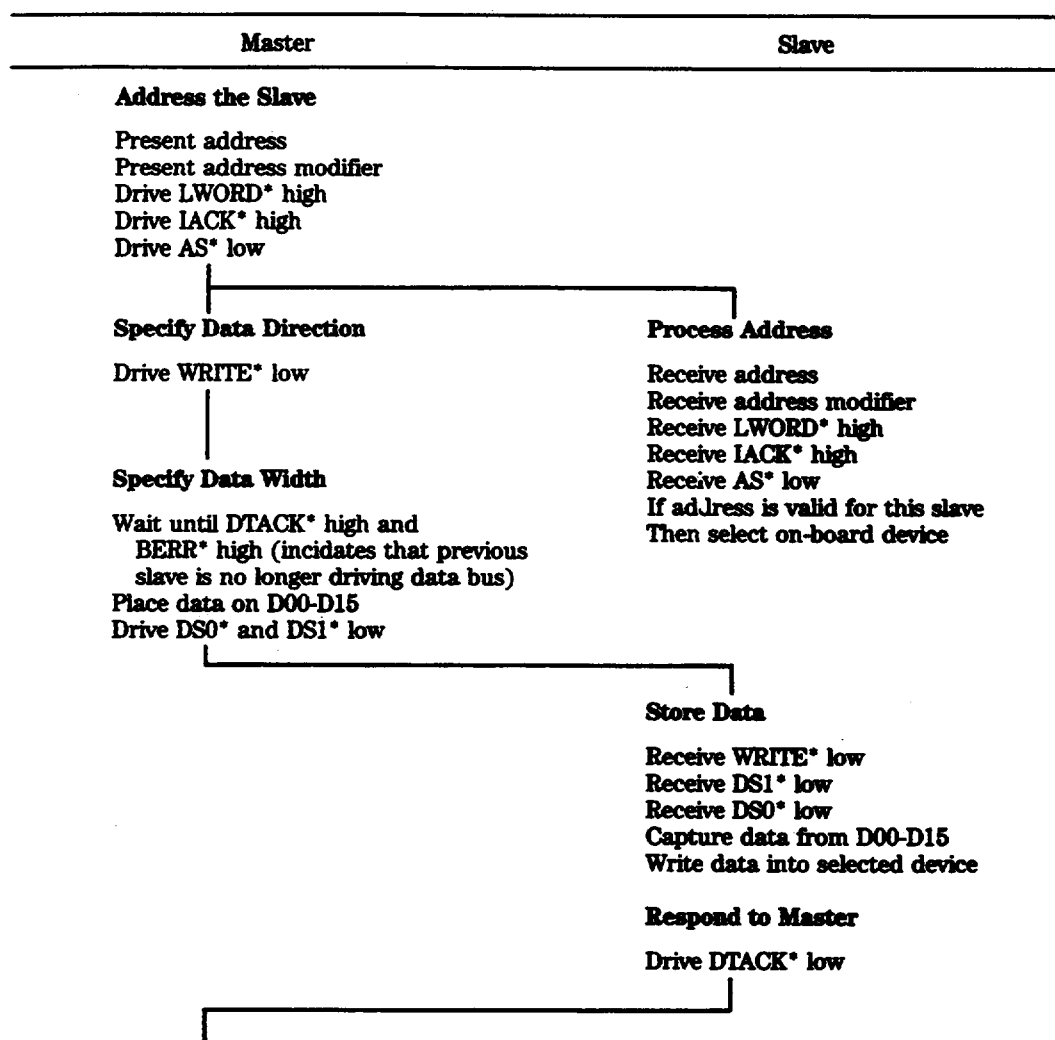


Figure 12—An Example of a Double-Byte Write Cycle

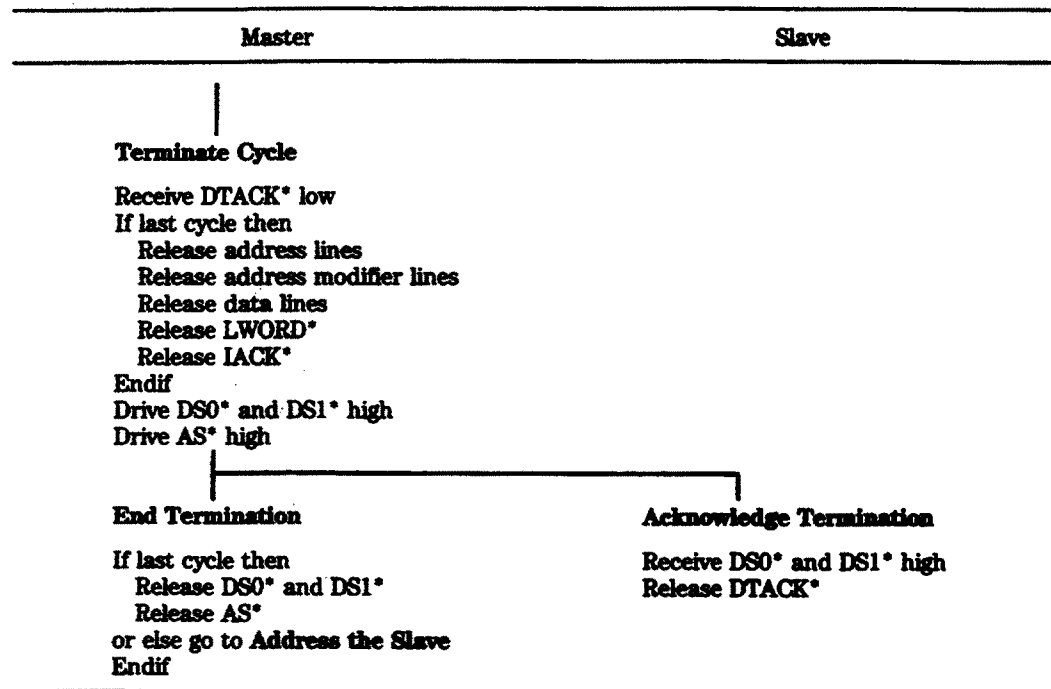


Fig 12 (Continued)—An Example of a Double-Byte Write Cycle

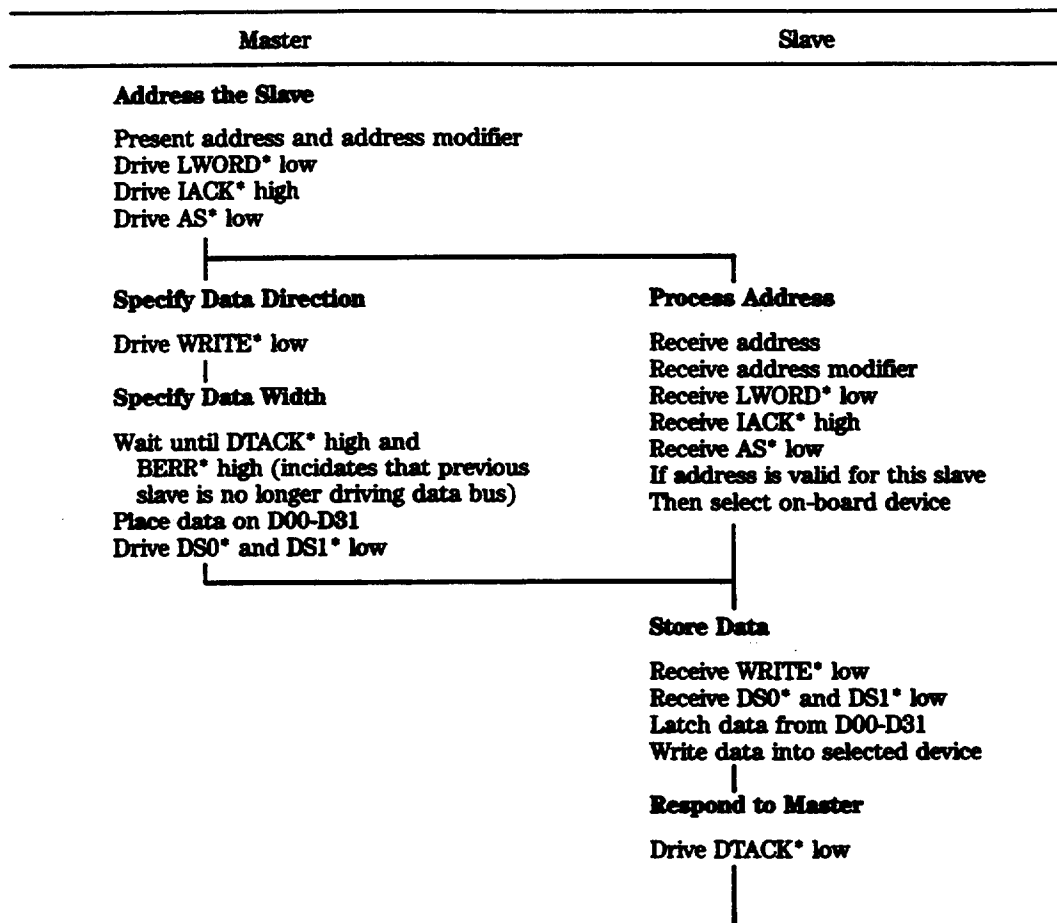


Figure 13—An Example of a Quad-Byte Write Cycle

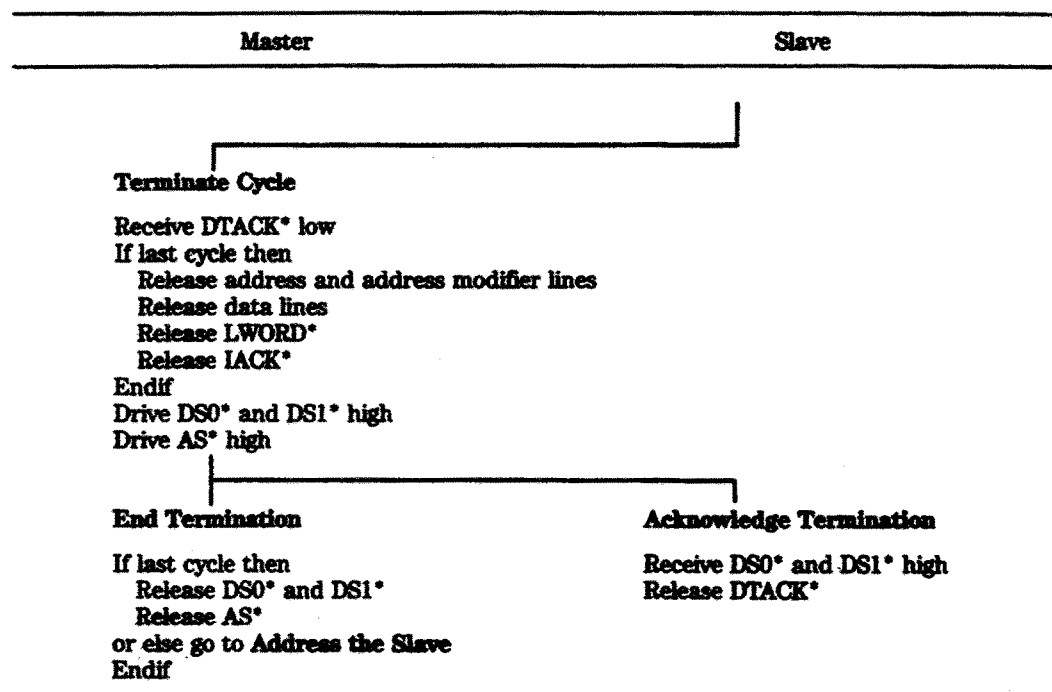


Fig 13 (Continued)—An Example of a Quad-Byte Write Cycle

2.4.2 Address Pipelining

Since separate strobes for the address and the data are defined, masters can broadcast the address for the next cycle while the data transfer for the previous cycle is still in progress. This is called *address pipelining*.

Permission 2.8 When the master detects that the responding slave has driven DTACK* or BERR* low, it may change the address, and, after driving AS* high for a minimum time, drive AS* low again.

For example, when a slave drives DTACK* low on a read cycle, the master is allowed to place a new address on the address bus while it is reading the data from the bus. This amounts to an overlapping of one cycle with the next one, and permits greater speeds on the bus.

Rule 2.18 All slaves shall be designed to accommodate address pipelining without loss of data or erroneous operation.

The following example offers two designs that comply with the requirements to accommodate address pipelining:

Observation 2.32 The responding slave might recognize its address and respond very quickly on the DTACK* or BERR* lines. Since the master is permitted to remove the address after the responding slave drives DTACK* or BERR* low, nonresponding slaves might not be able to decode the addressing information before the master removes it from the bus.

Observation 2.33 Because a master might broadcast a new address while a previous cycle is finishing, the designer of slave boards needs to ensure that the second assertion of AS* does not invalidate the first address when it is still needed by on-board logic to maintain the data on the bus.