

An American National Standard

IEEE Standard for a Simple 32-Bit Backplane Bus: NuBus

Sponsor

**Technical Committee on Microcomputers and Microprocessors
of the
IEEE Computer Society**

Approved June 11, 1987

IEEE Standards Board

Approved February 24, 1988

American National Standards Institute

ISBN 0471-62283-4

Library of Congress Catalogue Number 88-04595

© Copyright 1988 by

**The Institute of Electrical and Electronics Engineers, Inc
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.

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 1196-1987, IEEE Standard for a Simple 32-Bit Backplane Bus: NuBus.)

The computer bus structure defined by this document is a high-performance (37.5 megabyte per second) backplane bus that represents a distinctly minimalist approach. The goals of its developers have been to create a 32-bit backplane bus with the functions required for multiprocessor systems, while keeping the bus simple through a sparsity of mechanism. The straightforwardness of the bus is a direct result of the combined goals of avoiding bias toward a particular CPU type and bias toward a particular system architecture. NuBus,¹ the name given this bus by its originators at MIT, is unique in that it provides performance and functions found at the core of other 32-bit buses, but only provides the core.

The effort that resulted in this standard is the work of many individuals over several years' time. The start of the NuBus, from which 1196 was derived, was at the Massachusetts Institute of Technology (MIT). A paper by Profs. Steve Ward and Chris Terman, entitled "A New Approach to Personal Computers" [1979], described their concept of a computer workstation that was bus-based rather than centered around a particular microprocessor. MIT specified and prototyped the original "NuBus" system, and then worked with Western Digital Corporation (WDC) to reengineer MIT's basic concepts. Individuals from both MIT and WDC, including Steve Ward, Chris Terman, Rae McLellan, Jim Gula, Pete Vogt, Dave Godeau, Jeff Arnold, and George White, created the essence of the present NuBus in two weeks of intense joint effort at Cambridge in the Spring of 1981. From that beginning more than five years ago to the current specification, the project involved a significant effort by many more individuals. We have all learned that creating a well-specified bus requires much more effort than it initially seems. Dan Schneider at WDC and Richard Greenblatt of Lisp Machine Inc. made key technical contributions early in the bus's development.

In early 1983, the work being done at Western Digital was moved to Texas Instruments and was first presented to an IEEE Working Group (P896—Futurebus). Since the Futurebus activity had many goals in common with NuBus, it was hoped that they could emerge as "sister" buses with different low-level technology, but a common set of abstract functionality. As the Futurebus evolved, this became less likely, and the Microprocessor Standards Committee established a separate Study Group for this activity in July of 1984. This Study Group refined the bus, added some functions, and rewrote the specification. Meetings were held in Dallas, Boston, Palo Alto, San Diego, La Jolla, and New York, in order to provide wide participation. The members of the Working Group are listed below. Fritz Whittington has done the work of taking excellent minutes at our meetings and has been the draft editor of the specification. Gerry Laws has been responsible for the mailings, and has been acting chairman when needed.

Texas Instruments owns issued and pending patents that are claimed to relate to this standard. By publication of this standard, no position is taken with respect to the validity of this claim or of any patent rights in connection herewith. Texas Instruments has filed a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. Details may be obtained from the publisher or Texas Instruments.

At the time this document was approved, members of the Working Group were as follows:

George P. White, *Chair*

R. Gordon Cook
Mark Garetz
Richard Greenblatt
Ron Hochsprung
Richard Kalish
Rikki Kirzner

Gerry Laws
Rae McLellan
Gregory Papadopoulos
Dan Schneider
Dave Stewart
Michael G. Thompson

Jim Truchard
Eike Waltz
Steve Ward
Fritz Whittington

¹NuBus is a trademark of Texas Instruments Incorporated.

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

Andrew Allison	Tom Harkaway	Richard Rawson
Peter J. Ashenden	David Hawley	S. I. Samoylenko
Matt Biewer	David James	Sara L. Samuels
John Black	Laurel Kaleda	Michael Schrempp
Richard Boberg	R. L. Kalish	Bill Shields
Paul Borrill	Richard Karpinski	Michael Smolin
Bradley Brown	Hubert Kirrman	Robert Stewart
Clyde Camp	Doug Kraft	Subramanganesan
John D. Charlton	Tom Kurihara	Michael Teener
Steve Cooper	Glen Langdon	Scott Tetrick
Randy Davis	Gerry Laws	Eike Waltz
J. Robert Davis	Tom Leonard	Carl Warren
Shirish P. Deodhar	Gary Lyons	George White
Jim Dunlay	James Nebus	Fritz Whittington
Wayne Fischer	Gary Nelson	Tom Wicklund
Jim Flournoy	Deene Ogden	Andrew Wilson
Gordon Force	Tom Pittman	Anthony Winter
Martin Freeman	Shlomo Pri-Tal	
David Gustavson	P. Reghunathan	

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	Leslie R. Kerr	L. John Rankine
Dennis Bodson	Jack Kinn	John P. Riganati
Marshall L. Cain	Irving Kolodny	Gary S. Robinson
James M. Daly	Joseph L. Koepfinger*	Frank L. Rose
Stephen R. Dillon	Edward Lohse	Robert E. Rountree
Eugene P. Fogarty	John May	William R. Tackaberry
Jay Forster	Lawrence V. McCall	William B. Wilkens
Kenneth D. Hendrix	L. Bruce McClung	Helen M. Wood
Irvin N. Howell	Donald T. Michael*	

*Member emeritus

CLAUSE	PAGE
1. General	1
1.1 Scope	1
1.2 Objective	1
1.3 Purpose	1
1.4 Definitions	2
1.5 References	4
2. Overview	4
2.1 Bus Lines	4
2.2 Bus Operation	5
3. Specification	6
3.1 Protocol	6
3.2 Physical	25
3.3 Compliance	39
Annex (Informative)	42

An American National Standard

IEEE Standard for a Simple 32-Bit Backplane Bus: NuBus

1. General

1.1 Scope

This standard describes a computer backplane bus optimized for 32-bit transfers, multiprocessor operations, and simplicity. In brief, this is a synchronous (10 MHz), multiplexed, multimaster bus that provides a strictly fair arbitration mechanism. The only bus transfers are read and write (and block transfer versions of each of these) to a single 32-bit address space. Geographic slot addressing and nondaisy-chain arbitration scheme make system configuration simpler by eliminating switches and jumpers. This minimalist approach results in a conceptually straightforward bus with a small pin count (51 active signal lines). Figure 1 shows the major elements of a typical NuBus system.

1.2 Objective

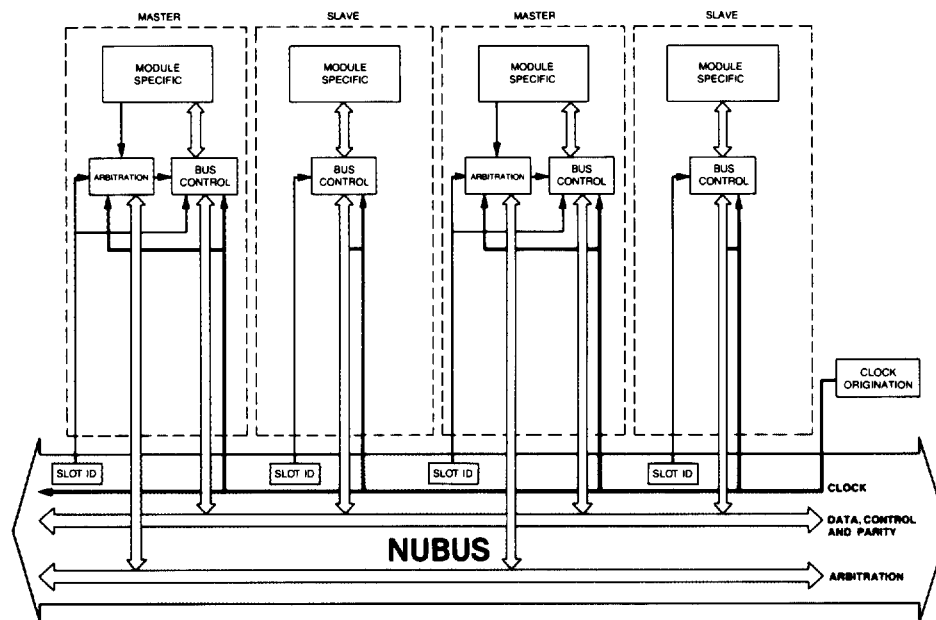
The objectives of this bus are:

- Optimized for 32-bit transfers
- System architecture independence
- Multiprocessor support
- Ease of system integration
- Sparsity of mechanism

These objectives result in a bus that is optimized for 32 bits, but simple enough for low-cost applications, such as 32-bit personal computers.

1.3 Purpose

This standard is intended to describe and specify the logical, electrical, and physical interface standard for circuit boards that allow them to connect to and communicate over a backplane. It also specifies the backplane environment that must be provided to these boards. This standard is oriented to designers of bus interface logic, designers of backplane environments, and those evaluating buses. In keeping with the minimalist philosophy of the bus, the standard has taken a similar approach. Section 2. provides an introduction to the bus, Section 3. is the “minimalist” core of the specification, and Appendix A describes implications and capabilities that follow from the rules presented in Section 3.



[Extracted from Texas Instruments NuBus™ specification document with permission of the publisher. Copyright (1983, 1986, 1988) Texas Instruments Incorporated.]

Figure 1— Simplified NuBus Diagram

Not specified by this standard are:

- Physical Environment—This includes topics such as how a backplane attaches to a rack, provisions for system cooling, resistance to vibration, etc.
- System Architecture—This is a low-level specification. Any system architectures (such as message passing protocols) are not within the scope of this standard.

1.4 Definitions

1.4.1 General

backplane: A circuit board with one or more bus connectors that provides signals for communication between bus modules, and provides certain resources to the connected modules.

board: A device connected to the bus. Usually constructed from a printed circuit board. Also referred to as a module.

bus: A set of signal lines to which a number of devices are connected and over which information is transferred between them.

byte: A set of 8 signals or bits taken as a unit.

half-word: For the purpose of this standard, a half-word is a 16-bit data item taken as a unit.

master: A bus device that initiates a transaction.

slave: A bus device that responds to a transaction.

word: For the purpose of this standard, a word is a 32-bit data item taken as a unit.

1.4.2 Protocol

ack cycle: A cycle in which a slave responds to a master and terminates a transaction.

arbitration: A collection of mechanisms that allow masters to access the bus without conflicting with each other.

arbitration contest: This is the core mechanism to resolve bus ownership between one or more competing masters. It takes two bus periods.

attention cycle: A single cycle in which a master indicates the start and acknowledge in the same cycle.

block transfer: A transaction in which a single address is conveyed by the master and multiple data items from sequential addresses are then communicated between the master and the slave.

bus lock: Method of a master ensuring continued tenure of the bus. Not identical to resource lock.

competitor: A master that participates in a particular arbitration contest.

cycle: One period of the bus clock, from rising edge to the next rising edge.

data cycle: A period in which data are valid and are acknowledged. This occurs when acknowledge is asserted at the end of a transaction and on intermediate acknowledges during a block transfer.

driving edge: A time corresponding to a rising edge of the bus clock.

fairness: A property of some arbitration techniques that ensures all modules will get access to the bus on approximately the same terms. This prevents modules from being “starved.”

null cycle: A type of an attention cycle that is used to dismiss a resource lock and initiate a re arbitration.

ownership: State of a master that has arbitrated and won the bus and has not yet lost a bus arbitration contest.

period: Time between two driving edges.

resource lock: A type of an attention cycle that indicates to slaves that data items will be referenced in a locked fashion and any nonbus path to referenced data items should be locked out.

sample edge: A time corresponding to a falling edge of the bus clock signal.

signal line: A conductor on the backplane other than ground, or power.

start cycle: A cycle that initiates a transaction. The address and transfer type are valid during this cycle.

tenure: Time period of unbroken ownership of the bus by a particular module. May consist of one or more transactions or attention cycles.

transaction: A sequence of cycles beginning with a start cycle and ending with an ack cycle that is used to convey data between a master and a slave.

1.4.3 Physical

asserted: The state of a signal line. Since all lines are active low signals, this state is the low state for all bus lines.

drive: A module activity causing a bus signal line to be in a particular state.

high, false, 1: Unasserted state of a bus line.

low, true, 0: Asserted state of a bus line.

open-collector: A type of bus driver (only drives low or not at all).

slot: A backplane location that accepts a NuBus module.

three-state: A type of bus driver. Either drives high, low, or not at all.

unasserted: The state of a signal line. Since all signal lines are active low, this state is the high state for all bus lines.

1.5 References

The following publications shall be used in conjunction with this standard.

- [1] ANSI/IEEE 1101-1987, IEEE Standard for Mechanical Core Specifications for Microcomputers.²
- [2] IEC 297-1-1986, Dimensions of Mechanical Structures of the 482.6 mm Series; Part 1: Panels and Racks.³
- [3] IEC 603-2-1980, Two-Part Connectors for Printed Boards, for Basic Grid of 2.54 mm (0.1 in), With Common Mounting Features.

2. Overview

This section introduces concepts required for a general understanding of the NuBus specification contained in Section 3. The material presented in this section of the standard does not *specify* the bus, but rather describes some characteristics of the bus. The description given in this section is deliberately brief and intuitive; Appendix A contains an in-depth elaboration of the specifications given in Section 3.

2.1 Bus Lines

NuBus lines can be grouped into four classes based on the functions they perform: utility signals, data transfer signals, arbitration signals, and power.

2.1.1 Utility Signals

The system clock and the bus timeout function are supplied by the terminator board, or other backplane module, and are not required on any NuBus board. Usually, the power fail warning and reset signals are generated by agents that do not reside on the backplane.

Clock (CLK^*) — Synchronizes bus arbitration and data transfers. Nominally 10 MHz with a duty cycle of 25%. Usually, bus signals are changed on the rising edge and sampled (75 ns later) on the falling edge.

Reset ($RESET^*$) — An open-collector signal, used to return all modules to their initial power-up state. May be asserted asynchronous to the CLK^* line.

Power Fail Warning (PFW^*) — This open-collector line signals that system power is about to fail. May be asserted asynchronous to the CLK^* line.

Card Slot Identification ($ID \langle 3 \dots 0 \rangle^*$) — These four lines are not bused, but are binary-encoded at each position to specify the module's position on the backplane.

Non-Master Request ($NMRQ^*$) — An asynchronous line asserted by boards that are not capable of becoming bus masters, to indicate a need for some service. The nature of this service and the determination of the provider are not specified by this standard.

²ANSI/IEEE publications can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018, or from the Institute of Electrical and Electronics Engineers, Service Center, Piscataway, NJ 08854-4150.

³IEC publications are available in the US from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018.

2.1.2 Bus Data Transaction Signals

These signals are all three-state lines, and include address/data, control, and parity lines.

Address and Data ($AD\langle 31\dots 0 \rangle^*$) — These lines are multiplexed to carry address information at the beginning of a transaction, and 8, 16, or 32 bits of data later in the transaction.

Transfer Mode ($TM\langle 1\dots 0 \rangle^*$) — At the beginning of the transaction, these two lines indicate the type of transaction being initiated. Later in the transaction, the responding module uses them to indicate success or failure of the requested transaction.

System Parity (SP^*) — This line carries the even parity bit generated from the 32 bits on the $AD\langle 31\dots 0 \rangle^*$ lines, if parity is being used.

System Parity Valid (SPV^*) — If asserted, it indicates that a parity bit has been generated for the $AD\langle 31\dots 0 \rangle^*$ lines.

Start Signal ($START^*$) — This signal is asserted at the start of a transaction, and also initiates an arbitration contest. Additionally, when asserted in conjunction with the ACK^* line, it denotes special nontransaction cycles called *attention cycles*.

Transfer Acknowledge (ACK^*) — The usual use of this signal is to indicate the ending cycle of a transaction. It has a special use if asserted during the same cycle with $START^*$.

2.1.3 Arbitration System Signals

The signals in this group are all open-collector lines and are used by the distributed arbitration logic to determine the next owner of the bus.

Bus Request ($RQST^*$) — This line is asserted by modules to indicate their desire to own the bus. The fair arbitration scheme guarantees that all modules requesting the bus will obtain ownership within some determinable maximum time.

Arbitration Signals ($ARB\langle 3\dots 0 \rangle^*$) — These four lines are bused and binary-encoded in the same manner as the $ID\langle 3\dots 0 \rangle^*$ lines. During an arbitration contest, contending modules compare these lines with the binary value of their own $ID\langle 3\dots 0 \rangle^*$ lines and drive the $ARB\langle 3\dots 0 \rangle^*$ lines according to the rules of the distributed arbitration logic. The net effect of an arbitration contest is that two cycles after starting a contest the $ARB\langle 3\dots 0 \rangle^*$ lines carry the binary-encoded number of the next bus owner.

2.1.4 Power Lines

Four voltages are defined for use by NuBus modules: +5 V, −12 V, −12 V, and −5.2 V. Voltage specifications such as required regulation are defined elsewhere in this standard; the amount of current available from each voltage supply is not specified by this standard.

2.2 Bus Operation

NOTE — The material presented in this section is descriptive and tutorial in nature, and is not a full specification of the bus.

2.2.1 Fundamental Concepts

- In general, signals are changed only on the rising edge of the system clock and sampled 75 ns later on the falling edge of the clock. The falling edge of the clock is called the *sample edge* and the rising edge of the clock is called the *driving edge*.

- The 100 ns time period between successive driving edges is called a *bus cycle* or simply a *cycle*. All cycles that exist can be categorized by considering the logic state of the following signals: $START^*$, ACK^* , $TM\langle 1\dots 0 \rangle^*$, and $AD\langle 1\dots 0 \rangle^*$.
- A *start cycle* is one in which $START^*$ is asserted (and ACK^* is not asserted). The $TM\langle 1\dots 0 \rangle^*$ and $AD\langle 1\dots 0 \rangle^*$ lines can be thought of as encoding one of sixteen possible kinds of transactions, eight of which are read transactions and the other eight are write transactions.
- An *ack cycle* is one in which ACK^* is asserted (and $START^*$ is not asserted). The $TM\langle 1\dots 0 \rangle^*$ lines encode one of four possible status codes, signifying successful completion, postponement, error, or bus timeout.
- An *attention cycle* is one in which both $START^*$ and ACK^* are asserted at the same time. the $TM\langle 1\dots 0 \rangle^*$ lines encode four possible types of attention cycles, two of which are defined as reserved for future use. The other two are used as broadcast messages to lock and unlock resources, and are beyond the scope of this overview.
- A *transaction* is the basic bus data transfer operation, which begins with a start cycle and ends with an ack cycle. Transactions may be categorized based on the category of the start cycle which initiates the transaction. Thus, there are read transactions, block write transactions, and so on.
- The bus is said to be *busy* for the time between a start cycle and its corresponding ack cycle. The bus is *idle* for the time between an ack cycle and the next start cycle.

2.2.2 Examples of Transactions

In order to use the bus, a module must first have ownership of the bus. A module (for example, the module in slot #7 on the backplane) obtains ownership by requesting the bus and waiting until the distributed arbitration logic determines that the next owner will be #7. Then, the module waits until the bus is idle, and does a start cycle to begin the transaction. If the module wishes to do a “read word” transaction, then the $AD\langle 1\dots 0 \rangle^*$ and $TM\langle 1\dots 0 \rangle^*$ lines will be encoded for this type of transaction, and the rest of the address lines will have the desired word address. If another module determines that the address refers to itself, then in some subsequent cycle it will place the requested data on the $AD\langle 31\dots 0 \rangle^*$ lines, and issue an ack cycle. This completes the transaction, and bus ownership may or may not pass to some other module, depending on the circumstances.

In a similar fashion, to write a word of data to another module on the bus, the start cycle would contain the appropriate code on the $TM\langle 1\dots 0 \rangle^*$ and $AD\langle 1\dots 0 \rangle^*$ lines, and the desired address in the rest of the address lines. The next cycle, the master would switch the $AD\langle 31\dots 0 \rangle^*$ lines to carry the 32 bits of data to be written. In the second or subsequent cycle, the address slave module would sample the data lines and then issue an ack cycle, thus completing the transaction.

Note that it is possible to have both read and write transactions be as short as two cycles, consisting of only the start cycle and the ack cycle. However, transactions may be longer than two cycles.

3. Specification

3.1 Protocol

3.1.1 Signal Determinacy

A signal line is determinate during a given cycle if it is in either an asserted or unasserted state within the specified setup and hold times of the system clock. When a signal is specified to be determinate, it shall be determinate as a consequence of one of the following:

- 1) If a signal is driven during cycle n , then it is determinate during cycle n .
- 2) If the three-state signal is driven asserted during cycle n and is not driven during cycles $n + 1$ and $n + 2$, then it is not guaranteed determinate during cycle $n + 1$, but is guaranteed (by the bus termination) to be unasserted during cycle $n + 2$.

- 3) If an open-collector signal is driven asserted during cycle n and is not driven during cycle $n + 1$, then it is guaranteed (by the bus termination) to be unasserted during cycle $n + 1$.
- 4) If a signal is unasserted during cycle n and is not driven during cycle $n + 1$, then it is guaranteed (by the bus termination) to remain unasserted during cycle $n + 1$.

3.1.2 Bus Cycles

A bus cycle is a single period of CLK^* , beginning with the rising edge. The rising edge of CLK^* is the driving edge, denoted by D_i , and is associated with the driving of bus signals during bus cycle i . The falling edge of CLK^* is the sample edge, denoted by S_i , and is associated with the sampling of bus signals during bus cycle i .

3.1.3 Transactions

A transaction consists of two or more bus cycles, the first of which is a start cycle and the last of which is an ack cycle. A start cycle is a bus cycle in which $START^*$ is asserted, ACK^* is unasserted, $AD\langle 31...0 \rangle^*$ carry an address, and $TM\langle 1... \rangle^*$ carry the transfer mode. An ack cycle is a bus cycle in which ACK^* is asserted, $START^*$ is unasserted. $AD\langle 31...0 \rangle^*$ carry data, and $TM\langle 1...0 \rangle^*$ carry the transfer response status.

3.1.3.1 Data Sizes

Three different data sizes may be transferred: bytes, halfwords, and words. The data transferred is unjustified as shown in Fig 2.

AD31*		AD0*	
Word			
Halfword 1		Halfword 0	
Byte 3	Byte 2	Byte 1	Byte 0

[Extracted from Texas Instruments NuBus™ specification document with permission of the publisher. Copyright (1983, 1986, 1988) Texas Instruments Incorporated.]

Figure 2— Relationship of Words, Halfwords, Bytes and $AD\langle 31...0 \rangle^*$ Lines

That is, a byte is conveyed on the same byte lane regardless of the transfer mode used to access it. Similarly, a halfword is conveyed on the same halfword lane regardless of the transfer mode used to access it. The data size and lane for a transaction is determined during a start cycle by the $TM\langle 1... \rangle^*$ and $AD\langle 1...0 \rangle^*$ lines as shown in Table 1.

NOTE — These encodings and bit and byte labels specify data transfers over the bus and do not necessarily directly correspond to the address encoding or byte labeling convention of a particular microprocessor.

Table 1— Transfer Mode Summary

$TM1^*$	$TM0^*$	$AD1^*$	$AD0^*$	Type of Cycle
L	L	L	L	Write byte 3
L	L	L	H	Write byte 2
L	L	H	L	Write byte 1
L	L	H	H	Write byte 0
L	H	L	L	Write halfword 1
L	H	L	H	Block write
L	H	H	L	Write halfword 0
L	H	H	H	Write word
H	L	L	L	Read byte 3
H	L	L	H	Read byte 2
H	L	H	L	Read byte 1
H	L	H	H	Read byte 0
H	H	L	L	Read halfword 1
H	H	L	H	Block read
H	H	H	L	Read halfword 0
H	H	H	H	Read word

During any cycle on which $AD\langle 31\dots \rangle^*$ are specified to carry data, all $AD\langle 31\dots 0 \rangle^*$ lines shall be determinate. This is independent of either the data size or lane.

NOTE — The state of any $AD\langle 31\dots 0 \rangle^*$ line not included in the transfer mode, although determinate, is unspecified.

3.1.3.2 Bus Parity

Parity checking of the $AD\langle 31\dots 0 \rangle^*$ lines is optional on a cycle by cycle basis. Thus parity is useful only when the module driving the $AD\langle 31\dots 0 \rangle^*$ lines generates parity and the module capturing the address or data checks it.

Two signals, SP^* and SPV^* are used to communicate the parity bit and to indicate parity is being used, respectively. SPV^* shall be determinate during any cycle in which the $AD\langle 31\dots 0 \rangle^*$ lines are specified to carry address or data.

If SPV^* is asserted during a cycle that $AD\langle 31\dots 0 \rangle^*$ carries an address or data, then SP^* shall be driven with the even parity of the $AD\langle 31\dots 0 \rangle^*$ lines. Even parity means that if an even number of $AD\langle 31\dots 0 \rangle^*$ lines are asserted, then SP^* is unasserted; otherwise SP^* is asserted.

If SPV^* is unasserted during a cycle that $AD\langle 31\dots 0 \rangle^*$ carries an address or data, then SP^* is unspecified and may be indeterminate.

Although byte and halfword transactions are supported, parity (if used) shall always be generated over the complete 32 bits of the $AD\langle 31\dots 0 \rangle^*$ lines. If parity errors are detected, the following applies:

During Address Cycle—A slave detecting a parity error shall not respond on this transaction.

During Data Cycle on Read—A master detecting a parity error shall complete the transaction. The data are presumed to be corrupted.

During Data Cycle on Write—An addressed slave detecting a parity error shall acknowledge with the error status code.

3.1.3.3 Acknowledgment

During an ack cycle the $TM\langle 1\dots 0 \rangle^*$ lines carry a transaction response status as shown in Table 2.

The transaction response status conveys information from the addressed slave to the master relevant to the current transaction. If the slave is going to respond, it shall complete the transaction with an ack cycle within 255 cycles following the start cycle.

NOTE — The master shall complete each transaction, regardless of the transaction response status. The action taken by a master in response to an abnormally completed transaction is not specified by this standard.

3.1.3.3.1 Bus Transfer Complete

This response indicates a normal valid completion of a bus transaction. If the transaction is a block transfer and bus transfer complete is issued without the transfer of any intermediate data, this response indicates that the slave does not support block transfers and that no data has been transferred. This is not an error condition.

Table 2— Transaction Response Status Summary

$TM1^*$	$TM0^*$	Type of Acknowledge
L	L	Bus transfer complete
L	H	Error
H	L	Bus timeout error
H	H	Try again later

3.1.3.3.2 Error

This response indicates an error condition was detected by the slave and indicates an unsuccessful transaction. During a read or block read an error response indicates that any data transferred may not be valid. During a write, event cycle, or block write, an error response indicates that the write may not have completed correctly.

3.1.3.3.3 Bus Timeout Error

This response indicates that no slave responded to the start cycle address. Timeout logic shall be implemented by the chassis in order to terminate the current transaction with a bus timeout error. If a period of 256 cycles have elapsed since a start cycle, and there has been no corresponding ack cycle, then the bus timeout logic shall assume the role of the slave, and shall generate an ack cycle with a bus timeout error code.

3.1.3.3.4 Try Again Later

This response indicates that the slave is unable to complete the transaction at this time. There is a strong implication that the transfer can be accomplished by some future request.

During a single read or write transaction, or the first data transfer cycle of a block transfer, this is not an error condition. The requesting module should retry the transaction. The maximum number of retries is not specified by this document, but slaves should not be designed in such a manner that a large number of retries is required to access them.

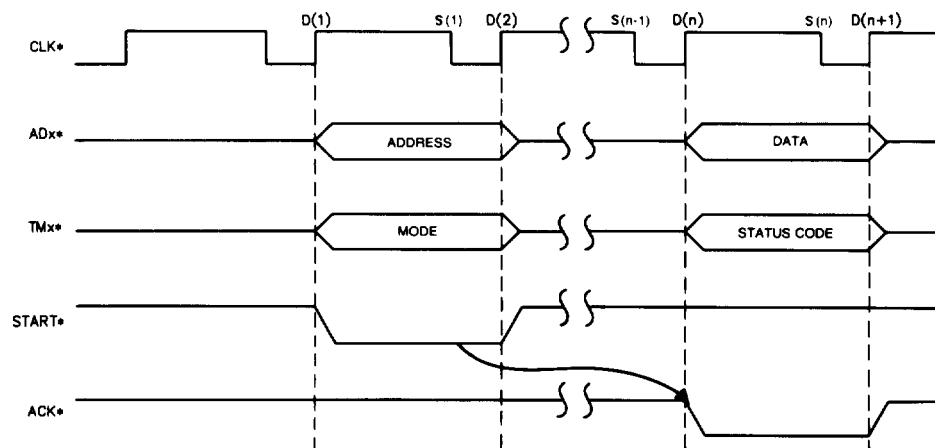
This response shall not occur during a block transfer on any data cycle other than the first one.

3.1.3.4 Single Data Cycle Transactions

A single data cycle transaction is a read transaction or a write transaction in which only a single data value is transferred.

3.1.3.4.1 Read Transactions

A read transaction consists of the following steps (refer to Fig 3):



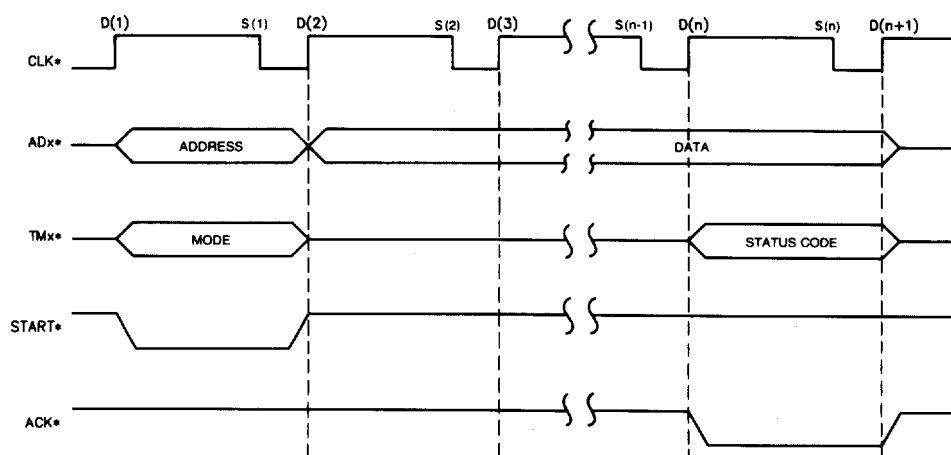
[Extracted from Texas Instruments NuBus™ specification document with permission of the publisher. Copyright (1983, 1986, 1988) Texas Instruments Incorporated.]

Figure 3— Read Transaction

- D_1 —The master asserts $START^*$ and drives the $AD\langle 31...0 \rangle^*$ lines with the desired address and drives the $TM\langle 1...0 \rangle^*$ lines with the appropriate transfer mode to initiate the transaction. The master shall ensure that ACK^* is unasserted.
- S_1 —Bus modules sample the $AD\langle 31...0 \rangle^*$ and $TM\langle 1...0 \rangle^*$ lines.
- D_2 —The master stops driving the $AD\langle 31...0 \rangle^*$, $TM\langle 1...0 \rangle^*$ and ACK^* lines. The master drives $START^*$ unasserted and waits for an ack cycle.
- D_n —The addressed slave drives the requested data onto the $AD\langle 31...0 \rangle^*$ lines, drives the appropriate transaction response status on the $TM\langle 1...0 \rangle^*$ lines, and asserts ACK^* . The slave shall ensure that $AD\langle 31...0 \rangle^*$ lines not indicated by the transfer mode are determinate.
- S_n —The bus master samples the $AD\langle 31...0 \rangle^*$ and $TM\langle 1...0 \rangle^*$ lines to receive the data and note any error condition.
- D_{n+1} —The addressed slave stops driving the $AD\langle 31...0 \rangle^*$, $TM\langle 1...0 \rangle^*$, and ACK^* lines. The bus owner shall drive ACK^* to a determinate state. This may be the D_1 of the next transaction.

3.1.3.4.2 Write Transactions

A write transaction consists of the following steps (refer to Fig 4):



[Extracted from Texas Instruments NuBus™ specification document with permission of the publisher. Copyright (1983, 1986, 1988) Texas Instruments Incorporated.]

Figure 4— Write Transaction

- D_1 — The master asserts $START^*$ and drives $AD\langle 31 \dots 0 \rangle^*$ lines with the desired address and the $TM\langle 1 \dots 0 \rangle^*$ lines with the appropriate transfer mode to initiate the transaction. The master shall ensure that ACK^* is unasserted.
- S_1 — Bus slaves sample the $AD\langle 31 \dots 0 \rangle^*$ and $TM\langle 1 \dots 0 \rangle^*$ lines.
- D_2-D_n — The master drives the data to be written onto the appropriate $AD\langle 31 \dots 0 \rangle^*$ lines. The master stops driving the $TM\langle 1 \dots 0 \rangle^*$ and ACK^* lines. The master drives $START^*$ unasserted and waits for an ack cycle. The determinacy requirement in this case indicates that the master drives all of the $AD\langle 31 \dots 0 \rangle^*$ lines independent of the data sizes or lanes.
- S_2-S_n — The addressed slave samples the $AD\langle 31 \dots 0 \rangle^*$ and $TM\langle 1 \dots 0 \rangle^*$ lines to receive the data. The data may be sampled on any cycle after the start cycle through the ack cycle.
- D_n — The addressed slave drives the appropriate transaction response status on the $TM\langle 1 \dots 0 \rangle^*$ lines and asserts ACK^* .
- D_{n+1} — The bus master stops driving the $AD\langle 31 \dots 0 \rangle^*$ lines, and the addressed slave stops driving the $TM\langle 1 \dots 0 \rangle^*$ and ACK^* lines. The bus owner shall drive ACK^* to a determinate state. This may be the D_1 of the next transaction.

3.1.3.4.3 Event Transactions

An event transaction is a special case of a write transaction that is used to post interrupts. The write address shall either specify a word, a halfword 0, or a byte 0, and the data written shall have $AD0^*$ asserted. The remaining AD^* lines required by the transfer mode should be either all asserted or all unasserted. Other combinations are reserved for future use. Any bus master can generate an interrupt for another module by performing an event transaction into an area of the address space that is being monitored by the slave module. Neither the method of monitoring nor the response elicited is specified by this standard.

3.1.4 Block Transfers

A block transfer is a read transaction or a write transaction in which multiple data values are transferred. A block transfer consists of a start cycle, multiple data cycles to or from sequential address locations, and an ack cycle. The number of data words transferred is controlled by the master and communicated during the start cycle. Allowed lengths of block transfers are two, four, eight, and sixteen words. Only word transfers are provided in block mode.