INTERNATIONAL STANDARD

ISO/IEC 15776

First edition
2001-12

VME64bus – Specification

Reference number
ISO/IEC 15776:2001(E)

INTERNATIONAL STANDARD

ISO/IEC 15776

First edition
2001-12

VME64bus – Specification

PRICE CODE XC

For price, see current catalogue

CONTENTS

VME64bus – SPECIFICATION

FOREWORD

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

2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

3) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 15776 was prepared by subcommittee 26: Microprocessor systems, of ISO/IEC joint technical committee 1: Information technology.

International Standards are drafted in accordance with ISO/IEC Directives, Part 3.

Annexes A, B and C form an integral part of this International Standard.

INTRODUCTION

The architectural concepts of VMEbus are based on the VERSAbus developed by Motorola in the late 1970s. Motorola's European Microsystems group in Munich, West Germany proposed the development of a VERSAbus-like product line of computers and controllers based on the Eurocard mechanical standard. To demonstrate the concept, Max Loesel and Sven Rau developed three prototype boards: (1) a 68000 CPU card, (2) a dynamic memory card, and (3) a static memory card. They named the new bus VERSAbus-E, which was later renamed "VME" by Lyman Hevle, then VP of the Motorola Microsystems Operation (and later the founder of VITA). VME is the acronym for VERSA-module Europe. Motorola, Mostek, and Signetics agreed to jointly develop and support the new bus architecture in early 1981.

John Black of Motorola, Craig McKenna of Mostek, and Cecil Kaplinsky of Signetics developed the first draft of the VMEbus specification. In October of 1981, at the Systems 81 trade show in Munich, West Germany, Motorola, Mostek, and Signetics announced their joint support for 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 (now VITA). 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 60297 standard, the formal specifications for Eurocard mechanical formats.

In the latter part of 1982, the French delegation of the International Electrotechnical Commission (IEC) proposed Revision B of the VMEbus specification 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 in the US. 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 Subcommittee.

The IEC, IEEE, and VMEbus Manufacturers Group (now VITA) distributed copies of Revision B for comment, and received requests for changes to the document as a result. These comments made it clear that it was time to go forward past revision B. In December of 1983, a meeting was held that included John Black, Mira Pauker, Wayne Fischer, and Craig McKenna. It was agreed that a revision C should be created, and that it should take into consideration all 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 VMEbus Manufacturers Group (now VITA) labeled the document Revision C.1 and placed it in the public domain. The IEEE labeled it P1014 Draft 1.2, and the IEC labeled it IEC 60821 Bus. Subsequent ballots in the IEEE P1014 group and in the MSC resulted in more comments, and required that the IEEE P1014 draft be updated. This work resulted in the ANSI/IEEE 1014-1987 specification.

In 1989, John Peters of Performance Technologies, Inc. (Rochester, NY) developed the initial concept of VME64: multiplexing address and data lines (A64/D64) on the VMEbus. This concept was shown for the first time in 1989 and placed in the VITA Technical Committee in 1990 as a performance enhancement to the VMEbus specification. In 1991, the PAR (Project Authorization Request) for P1014R (revisions to the VMEbus specification) was granted by the IEEE. Ray Alderman, Technical Director of VITA, co-chaired the activity with Kim Clohessy of DY 4 Systems (Nepean, Ontario, Canada).

At the end of 1992, the additional enhancements to VMEbus (A40/D32, Locked Cycles, Rescinding DTACK*, Autoslot-ID, Auto System Controller, and enhanced DIN connector mechanicals) required more work to complete this document. In 1992, the VITA Technical Committee suspended work with the IEEE and sought accreditation as a standards developer organization (SDO) with the American National Standards Institute. The original IEEE Par P1014R was subsequently withdrawn by the IEEE. The VITA Technical Committee returned to using the public domain VMEbus C.1 specification as its base level document to which it added new enhancements. This enhancement work was undertaken entirely by the VITA Technical Committee resulting in this document. The tremendous undertaking of the document editing was accomplished by Kim Clohessy of DY 4 Systems, the technical co-chair of the activity with great help from Frank Hom who created the mechanical drawings, and with exceptional contributions by each chapter editor.

Additional enhancements proposed to the VME64 Subcommittee have been placed in another VITA subcommittee: the VME64 Extensions Document. Two other activities began in late 1992: (1) BLLI (VMEbus Board-level Live Insertion Specifications), and (2) VSLI (VMEbus System-level Live Insertion with Fault Tolerance).

New activities begun in 1993 using the base-VME architecture involve the implementation of high-speed serial and parallel sub-buses for use as I/O interconnections and data mover subsystems. These architectures can be used as message switches, routers, and small multiprocessor parallel architectures.

VITA's application for recognition as an accredited standards developer organization of ANSI (American National Standards Institute) was granted in June 1993. Numerous other documents, including mezzanine, P2, and serial bus standards, have been placed with VITA as the Public Domain Administrator of these technologies.

VMEbus Specification Genealogy

VMEbus	Revision B and C.1 (Public Domain)
IEEE 1014-1987	Versatile Backplane Bus: VMEbus
VITA 1-1994	VME64 Specification
IEEE 1096-1988	VSBbus Specification (IEEE)
IEC 60821:1991	VMEbus – Microprocessor system bus for 1 byte to 4 byte data
IEEE 1101.1	IEEE Standard for Mechanical Core Specifications for Microcomputers Using IEC 60603-2 Connectors
IEEE 1101.2	IEEE Standard for Mechanical Core Specification for Conduction-Cooled Eurocards

This standard was constructed through the many hours of hard work by the members of the VME64 Subcommittee (of the VITA Technical Committee) and the commitment of their companies to this standard.

NAME	COMPANY
Ray Alderman	PEP Modular Computers
Michael Humphrey	VITA
John Rynearson	VITA
Martin Blake	BICC-VERO UK
Kim Clohessy	DY 4 Systems, Inc.
Jing Kwok	DY 4 Systems, Inc.
Clarence Peckham	Heurikon Corporation
Dennis Terry	Heurikon Corporation
Wayne Fischer	Force Computers Inc.
Jack Regula	Consultant
Tad Kubic	Dawn VME Products, Inc.
Will Hamsher	AMP, Inc.
Doug Reubendall	AMP, Inc.
William Mahusen	Performance Technologies, Inc.
Thanos Mentzelopoulos	Ironics, Inc.
Joel Silverman	Radstone Technology Corp.
Colin Davies	Radstone Technology UK
Frank Hom	Electronic Solutions
Keith Burgess	Mizar
John Black	Micrology PBT
Greg Hill	Hewlett-Packard Co.
Sam Babb	Hewlett Packard Co.
Ben LaPointe	Motorola GEG
Chau Pham	Motorola MCG
Mac Rush	Motorola MCG
Mike Hasenfratz	Micro Memory, Inc.
Richard DeBock	Matrix Corp.
Richard O'Connor	Newbridge Microsystems
Michael Bryant	PEP Modular Computers
Dr. Chris Eck	CERN-Geneva
Mike Thompson	Schroff, Inc.
Eike Waltz	Schroff, Inc.
Tom Baillio	Mercury Computer Systems, Inc.
Steve Corbesero	Mupac Corp.
Dave Horton	Cypress Semiconductor
Mike Maas	Cypress Semiconductor
Mike Munroe	Hybricon Corp.

The following are acknowledged for their extra contributions as chapter editors.

Kim Clohessy	DY-4 Systems Inc.
Wayne Fischer	Force Computers, Inc.
Chau Pham	Motorola MCG
Frank Hom	Electronic Solutions
Richard DeBock	Matrix Corp.
Jing Kwok	DY 4 Systems, Inc.
Mike Hasenfratz	Micro Memory, Inc.

CANVASS BALLOT

Consensus for this standard was achieved by use of the Canvass Method.

The following organizations, recognized as having an interest in the standardization of VME64, participated in the Canvass Ballot process. Inclusion in this list does not necessarily imply that the organization concurred with the submittal of the proposed standard to ANSI.

767 AWACS
Adept Technology
AMP
AT&T Bell Laboratories
Berg Electronics
Bit 3 Computer
CERN
CSPI
Cypress Semiconductor
Dawn VME Products
Dialogic Corporation
Digital Equipment Corp.
DY 4 Systems
Electronic Solutions
Force Computers
Harting Elektronik
Heurikon Corporation
Hewlett-Packard
Hughes Aircraft Company
Hybricon Corporation
IBM
IXTHOS
Loral Western Devel. Lab
Los Alamos Nat'l Lab
Matrix Corporation
Micro Memory
Micrology
MITRE Corporation
Motorola Computer Group
Mupac Corporation
Newbridge Microsystems
Object Technology Inc.
PEP Modular Computers
Philips Ind. Automation
Picosoft
Radstone Technology
Schroff
Technology Consulting
Texas Instruments
VERO Electronics
VITA
VME MEMBER
Winchester Electronics

VME64bus – SPECIFICATION

1 General

1.1 Scope and object

The VMEbus specification defines an interfacing system used to interconnect microprocessors, data storage, and peripheral control devices in a closely coupled hardware configuration. The system has been conceived with the following objectives:

a) to allow communication between devices on the VMEbus without disturbing the internal activities of other devices interfaced to the VMEbus;

b) to specify the electrical and mechanical system characteristics required to design devices that will reliably and unambiguously communicate with other devices interfaced to the VMEbus;

c) to specify protocols that precisely define the interaction between the VMEbus and devices interfaced to it;

d) to provide terminology and definitions that describe the system protocol;

e) to allow a broad range of design latitude so that the designer can optimize cost and/or performance without affecting system compatibility;

f) to provide a system where performance is primarily device limited, rather than system interface limited.

1.2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60297-1:1986, *Dimensions of mechanical structures of the 482,6 mm (19 in) series – Part 1: Panels and racks*

IEC 60297-2:1982, *Dimensions of mechanical structures of the 482,6 mm (19 in) series – Part 2: Cabinets and pitches of rack structures*

IEC 60297-3:1984, *Dimensions of mechanical structures of the 482,6 mm (19 in) series – Part 3: Subracks and associated plug-in units*

IEC 60297-4:1995, *Mechanical structures for electronic equipment – Dimensions of mechanical structures of the 482,6 mm (19 in) series – Part 4: Subracks and associated plug-in units – Additional dimensions*

IEC 60603-2:1995, *Connectors for frequencies below 3 MHz for use with printed boards – Part 2: Detail specification for two-part connectors with assessed quality, for printed boards, for basic grid of 2.54 mm (0.1 in) with common mounting features*

IEC 61076 (all parts), *Connectors with assessed quality, for use in d.c., low frequency analogue and digital high speed data applications*

IEEE 1101.2, *Standard for Mechanical Core Specifications for Conduction-Cooled Eurocards*

IEEE 1394, *Standard for a High Performance Serial Bus*

1.3 VMEbus interface system elements

1.3.1 Basic definitions

The VMEbus structure can be described from two points of view: its mechanical structure and its functional structure. The mechanical specification describes the physical dimensions of subracks, backplanes, front panels, plug-in boards, etc. The VMEbus functional specification describes how the bus works, what functional modules are involved in each transaction, and the rules which govern their behavior. The following informal definitions describe some basic terms used for both the mechanical and the functional structure of the VMEbus.

1.3.1.1 VMEbus mechanical structure

1.3.1.1.1
VMEbus backplane
printed circuit (PC) board with 96 or 160 pin connectors and signal paths that bus the connector pins

Some VMEbus systems have a single PC board, called the J1 backplane. It provides the signal paths needed for basic operation. Other VMEbus systems also have an optional second PC board, called a J2 backplane. It provides the additional 96 or 160 pin connectors and signal paths needed for wider data and address transfers. Still others have a single PC board that provides the signal conductors and connectors of both the J1 and J2 backplanes.

1.3.1.1.2
board
printed circuit (PC) board, its collection of electronic components, with either one or two 96 or 160 pin connectors that can be plugged into VMEbus backplane connectors

1.3.1.1.3
slot
position where a board can be inserted into a VMEbus backplane

If the VMEbus system has both a J1 and a J2 backplane (or a combination J1/J2 backplane) each slot provides a pair of 96 or 160 pin connectors. If the system has only a J1 backplane, then each slot provides a single 96 or 160 pin connector.

1.3.1.1.4
subrack
rigid framework that provides mechanical support for boards inserted into the backplane, ensuring that the connectors mate properly and that adjacent boards do not contact each other. It also guides the cooling airflow through the system, and ensures that inserted boards do not disengage themselves from the backplane due to vibration or shock

1.3.1.2 VMEbus functional structure

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

1.3.1.2.1
backplane interface logic
special interface logic that takes into account the characteristics of the backplane: its signal line impedance, propagation time, termination values, etc.

The VMEbus specification prescribes certain rules for the design of this logic based on the maximum length of the backplane and its maximum number of board slots.

1.3.1.2.2
functional module
collection of electronic circuitry that resides on one VMEbus board and works together to accomplish a task

1.3.1.2.3
data transfer bus
one of the four buses provided by the VMEbus backplane

The Data Transfer Bus allows Masters to direct the transfer of binary data between themselves and Slaves. (Data Transfer Bus is often abbreviated DTB.)

1.3.1.2.4
data transfer bus cycle
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, the address broadcast and then zero or more data transfers. There are 34 types of Data Transfer Bus cycles. They are defined later in this chapter.

1.3.1.2.5
master
functional module that initiates DTB cycles in order to transfer data between itself and a Slave module

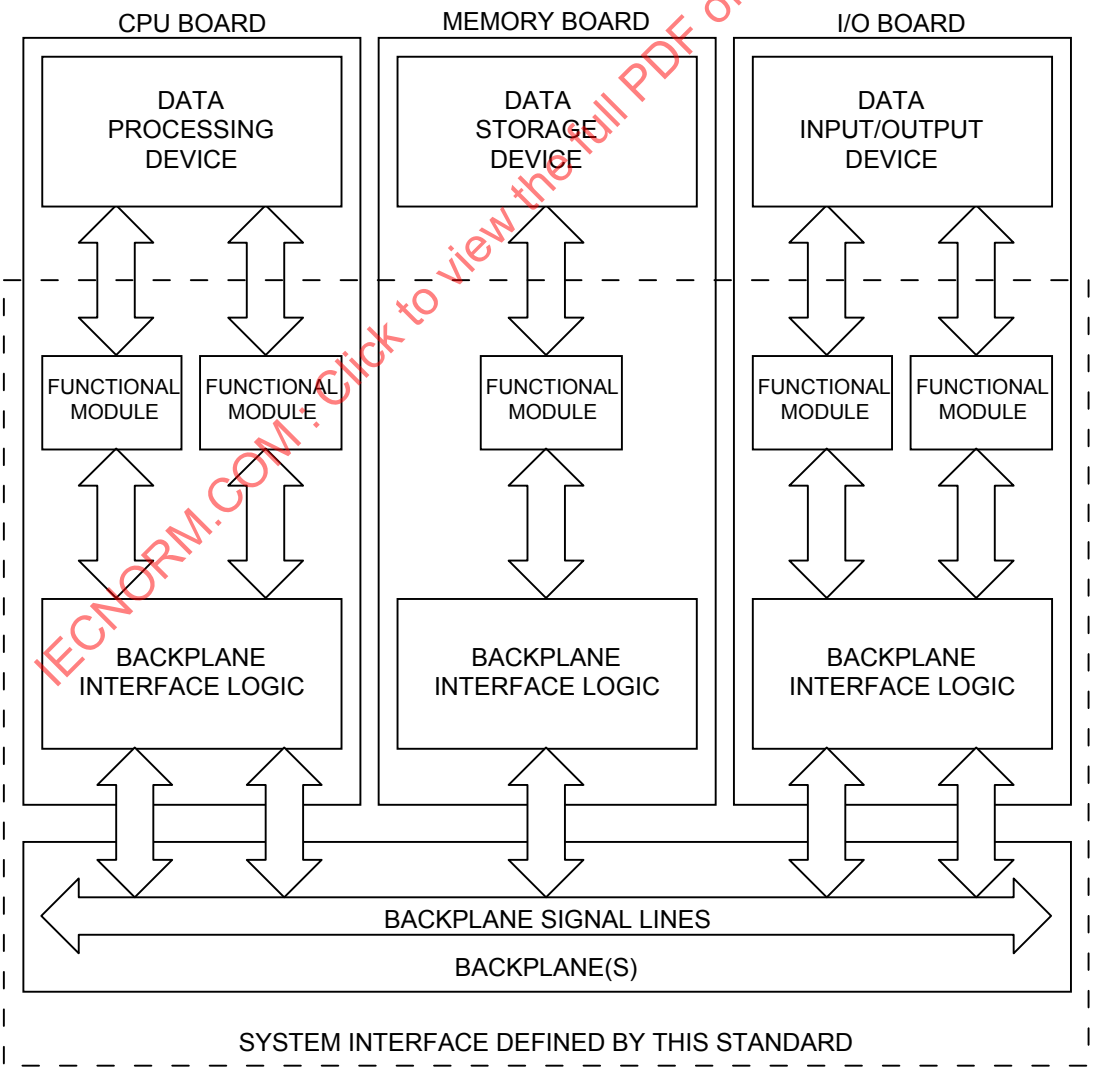


Figure 1 – System elements

1.3.1.2.6
slave
a functional module that detects DTB cycles initiated by a Master and, when those cycles specify its participation, transfers data between itself and the Master

1.3.1.2.7
location monitor
a functional module that monitors data transfers over the DTB in order 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

1.3.1.2.8
bus timer
a functional module with a preset time-out period which terminates the DTB cycle if a transfer exceeds the time-out period. Without this module, if the Master tries to transfer data to or from a nonexistent Slave location it might wait forever. The Bus Timer prevents this by terminating the cycle

1.3.1.2.9
priority interrupt bus
one of the four buses provided by the VMEbus backplane

The Priority Interrupt Bus allows Interrupter modules to send interrupt requests to Interrupt Handlers.

1.3.1.2.10
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

1.3.1.2.11
interrupt handler
a functional module that detects interrupt requests generated by Interrupters and responds to those requests by asking for Status/ID information.

1.3.1.2.12
status/ID
an eight, sixteen, or thirty-two-bit value returned by an interrupter to an interrupt handler during an interrupt acknowledge cycle

1.3.1.2.13
daisy-chain
a special type of VMEbus signal line that is used to propagate a signal level from board to board, 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 VMEbus.

1.3.1.2.14
IACK daisy-chain driver
a functional module which activates the interrupt acknowledge daisy-chain whenever an Interrupt Handler acknowledges an interrupt request

This daisy-chain ensures that only one Interrupter will respond with its STATUS/ID when more than one has generated an interrupt request.

1.3.1.2.15
arbitration bus
one of the four buses provided by the VMEbus backplane

This bus allows an Arbiter module and several Requester modules to coordinate use of the DTB.

1.3.1.2.16
requester
a functional module that resides on the same board as a Master or Interrupt Handler and requests use of the DTB whenever its Master or Interrupt Handler needs it

1.3.1.2.17
arbiter
a functional module that accepts bus requests from Requester modules and grants control of the DTB to one Requester at a time

1.3.1.2.18
utility bus
one of the four buses provided by the VMEbus backplane

This bus includes signals that provide periodic timing and coordinate the power-up and power-down of VMEbus systems.

1.3.1.2.19
CR/CSR
a functional module that provides Configuration ROM information and Control and Status Registers. The module provides manufacturing and board ID and other important board information

The CSRs are used for software configuration of a VMEbus system.

1.3.1.2.20
system clock driver
a functional module that provides a 16 MHz timing signal on the Utility Bus

1.3.1.2.21
serial bus
a functional module that provides a bused 2 wire interface between cards in the backplane, independent of the other VMEbus modules

1.3.1.2.22
power monitor module
a functional module that monitors the status of the primary power source to the VMEbus system, and signals when that power has strayed outside the limits required for reliable system operation

Since most systems are powered by an a.c. source, the Power Monitor is typically designed to detect drop-out or brown-out conditions on AC lines.

1.3.1.2.23
system controller board
a board which resides in slot 1 of a VMEbus backplane and has a System Clock Driver, an Arbiter, an IACK Daisy-Chain Driver, and a Bus Timer; some also have a Power Monitor

1.3.1.3 Types of cycles on the VMEbus

1.3.1.3.1
read cycle
a DTB cycle used to transfer 1, 2, 3, 4 or 8 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 modifier and address and checks to see if it is to respond to the cycle. If so, it retrieves the data from its internal storage, places it on the data bus and acknowledges the transfer. The Master then terminates the cycle.

1.3.1.3.2
write cycle
a DTB cycle used to transfer 1, 2, 3, 4 or 8 bytes from a Master to a Slave

The cycle begins when the Master broadcasts an address and address modifier and places data on the DTB. Each Slave captures the address and address modifier and checks to see if it is to respond to the cycle. If so, it stores the data and then acknowledges the transfer. The Master then terminates the cycle.

1.3.1.3.3
block read cycle
a DTB cycle used to transfer a block of 1 to 256 bytes from a Slave to a Master

This transfer is done using a string of 1, 2, or 4 byte data transfers. Once the block transfer is started, the Master does not release the DTB until all of the bytes have been transferred. It differs from a string of read cycles in that the Master broadcasts only one address and address modifier (at the beginning of the cycle). Then the Slave increments this address on each transfer so that the data for the next transfer is retrieved from the next higher location.

1.3.1.3.4
block write cycle
a DTB cycle used to transfer a block of 1 to 256 bytes from a Master to a Slave. The block write cycle is very similar to the block read cycle. It uses a string of 1, 2, or 4 byte data transfers. The Master does not release the DTB until all of the bytes have been transferred. It differs from a string of write cycles in that the Master broadcasts only one address and address modifier (at the beginning of the cycle). Then the Slave increments this address on each transfer so that the data from the next transfer is stored in the next higher location.

1.3.1.3.5
multiplexed cycle
a DTB cycle that transfers address information and/or data information using both the address and data buses. Multiplexed cycles are used in four cases.

a) A64 – the full address bus and the full data bus are combined to create a 64 bit address.

b) MBLT – the full address bus and the full data bus are combined to create a 64 bit data word.

c) A40 – The full 24 bit address bus and the full 16 bit data bus on the P1/J1 connector are combined to create a 40 bit address. This mode is especially useful for 3U boards which have a J1 connector only.

d) MD32 – The lower 16 address lines and the lower 16 data lines are combined to create a 32 bit data word. This mode is especially useful for 3U modules.

Multiplexed cycles are used in both basic transfers and block transfers. A64 and A40 basic transfers support 1 byte, 2 byte and 4 byte transfers. In addition MBLT cycles support 8 byte transfers.

A Multiplexed Cycle will have an Address phase that is separate from the Data phase. The Address phase may include (i.e. A64 and A40 cycles) or may not include (i.e. A32, A24 cycles) the use of the Data bus. The Data phase may include (MBLT, MD32 cycles) or may not include (i.e. D32, D16, D08(OE) cycles) the use of the Address bus.

Multiplexed block transfers can have 1 to 256 transfers between the Master and a Slave. With 8 byte transfers, up to 2,048 bytes can be transferred in one block.

1.3.1.3.6
read-modify-write cycle
a 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.

1.3.1.3.7
address-only cycle
a DTB cycle that consists of an address broadcast, but no data transfer

Slaves do not acknowledge ADDRESS-ONLY cycles and Masters terminate the cycle without waiting for an acknowledgment. No data strobes or acknowledge strobes are asserted in an ADDRESS-ONLY cycle.

1.3.1.3.8
address-only-with-handshake cycle
a DTB cycle that consists of an address broadcast, but no data transfer

The addressed Slave responds in the same manner as a standard access cycle.

1.3.1.3.9
interrupt acknowledge cycle
a DTB cycle, initiated by an Interrupt Handler, which 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.

1.3.2 Basic VMEbus structure

The VMEbus interface system consists of backplane interface logic, four groups of signal lines called 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 defined in this document 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 which interacts with the VMEbus in the manner described, but uses different on-board signals, or monitors other VMEbus signals. VMEbus boards might be designed to include any combination of the functional modules defined by this standard.

The VMEbus functional structure can be divided into four categories. Each consists of a bus and its associated functional modules which work together to perform specific duties. Figure 2 shows the VMEbus functional modules and buses. Each category is briefly summarized below.

1.3.2.1
data transfer
devices transfer data over the Data Transfer Bus (DTB), which contains data and address pathways and associated control signals

Functional modules called Masters, Slaves, Interrupters, and Interrupt Handlers use the DTB to transfer data between each other. Two other modules, called Bus Timer and IACK Daisy-Chain Driver also assist them in this process.

1.3.2.2

DTB arbitration
since a VMEbus system can be configured with more than one Master or Interrupt Handler, a means is provided to transfer control of the DTB between them in an orderly manner and to guarantee that only one Master controls the DTB at a given time. The Arbitration Bus modules (Requesters and Arbiter) coordinate the control transfer

1.3.2.3
priority interrupt
the priority interrupt capability of the VMEbus provides a means by which devices can request services 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.

1.3.2.4
utilities
periodic clocks, initialization, and failure detection are provided by the Utility Bus

It includes a general purpose system clock line, a system reset line, a system fail line, an a.c. fail line, and two serial lines. Utilities also include power and ground pins.

1.4 VMEbus specification diagrams

As aids to defining or describing VMEbus operation, several types of diagrams are used, including:

a) Timing diagrams that show the timing relationships between signal transitions. The times involved will have minimum and/or maximum limits associated with them. Some of the times specified on these diagrams specify the behavior of the backplane interface logic, while other times specify the interlocked behavior of the functional modules.

b) Sequence diagrams that are similar to timing diagrams but show only the interlocked timing relationships of the functional modules. These diagrams are intended to show a sequence of events, rather than to specify the times involved. For example, a sequence diagram might indicate that module A cannot generate signal transition B until it detects module C's generation of signal transition D.

c) Flow diagrams that show a stream of events as they would occur during a VMEbus operation. The events are stated in words and result from interaction of two or more functional modules. The flow diagram describes VMEbus operations in a sequential manner and, at the same time, shows interaction of the functional modules.

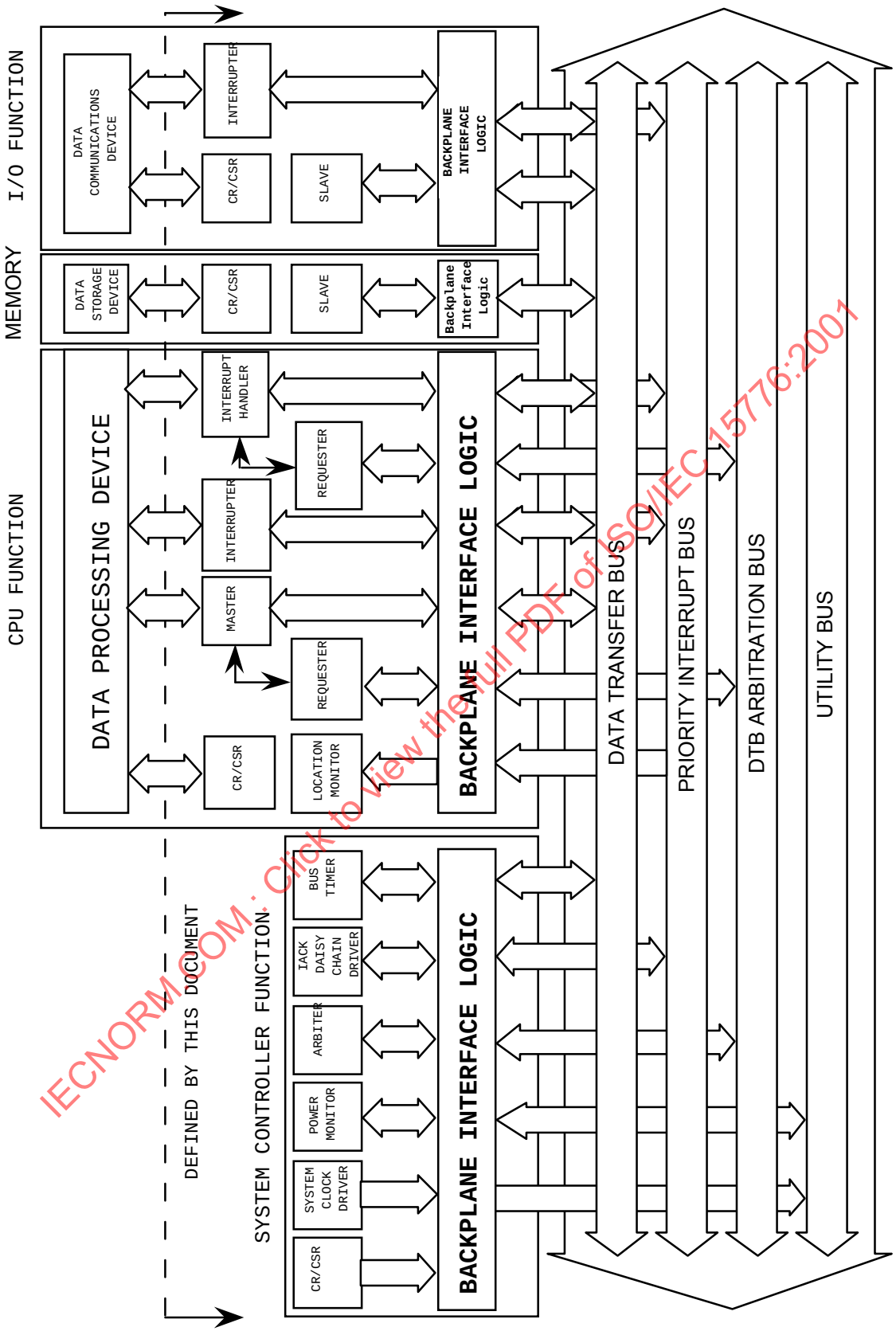


Figure 2 – Functional modules and buses

1.5 Specification terminology

To avoid confusion, and to make very clear what the requirements for compliance are, many of the paragraphs in this document are labelled with keywords that indicate the type of information they contain. The keywords are listed below

RULE
RECOMMENDATION
SUGGESTION
PERMISSION
OBSERVATION

Any text not labelled with one of these keywords describes the VMEbus structure or operation. It is written in either a descriptive or a narrative style. These keywords are used as follows:

RULE chapter.number

Rules form the basic framework of the VMEbus specification. They are sometimes expressed in text form and sometimes in the form of Figures, tables, or drawings. All VMEbus rules MUST be followed to ensure compatibility between VMEbus designs. Rules are characterized by an imperative style. The upper-case words MUST and MUST NOT are reserved exclusively for stating rules in this document and are not used for any other purpose.

RECOMMENDATION chapter.number

Wherever a recommendation appears, designers would be wise to take the advice given. Doing otherwise might result in some awkward problems or poor performance. While the VMEbus has been designed to support high performance systems, it is possible to design a VMEbus system that complies with all the rules, but has abysmal performance. In many cases, a designer needs a certain level of experience with the VMEbus in order to design boards that deliver top performance. Recommendations found in this document are based on this kind of experience and are provided to designers to speed their traversal of the learning curve.

SUGGESTION chapter.number

In the VMEbus specification, a suggestion contains advice which is helpful but not vital. The reader is encouraged to consider the advice before discarding it. Some design decisions that need to be made in designing VMEbus boards are difficult until experience has been gained with the VMEbus. Suggestions are included to help a designer who has not yet gained this experience. Some suggestions have to do with designing boards that can be easily reconfigured for compatibility with other boards, or with designing the board to make the job of system debugging easier.

PERMISSION chapter.number

In some cases a VMEbus rule does not specifically prohibit a certain design approach, but the reader might be left wondering whether that approach might violate the spirit of the rule, or whether it might lead to some subtle problem. Permissions reassure the reader that a certain approach is acceptable and will cause no problems. The upper-case word MAY is reserved exclusively for stating permissions in this document and is not used for any other purpose.

OBSERVATION chapter.number

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

1.5.1 Signal line states

The VMEbus specification describes its protocol 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. (See clause 6 for voltage thresholds used on the VMEbus.)

There are two possible transitions which can appear on a signal line, and these are called “edges”. A rising edge is the time during which a signal level makes its transition from a low level to a high level. The falling edge is the time during which a signal level 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. The problem with doing this is that board 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 if designers know what the maximum and minimum loading will be, they still need to spend time in the lab, experimenting to find out which drivers will provide the needed rise and fall times.

In fact, 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. In order to trade off all of these factors the board designer would have to study transmission line theory, as well as certain specific parameters of drivers and receivers which are not normally found in most manufacturers' data sheets.

Recognizing all of this, the VMEbus standard doesn’t specify rise and fall times. Instead, it specifies the electrical characteristics for drivers and receivers and specifies the backplane design. It also tells designers how the worst case bus loading will affect the propagation delay of these drivers so that they can ensure that the VMEbus timing is met before building a board. If VMEbus designers follow these propagation delay guidelines, their boards will operate reliably with other VMEbus compatible boards under worst case conditions.

System performance is influenced by the rise time of one of the open collector control signals. To improve the performance of the system, the concept of a rescinding signal has been introduced. A rescinding signal is an open collector type output that is initially driven high and then released back to open collector mode within a short period of time.

1.5.2 Use of the asterisk (*)

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

a) An asterisk (*) following the signal name of signals which are level significant denotes that the signal is true or valid when the signal is low.

b) An asterisk (*) following the signal name of signals which are edge significant denotes that the actions initiated by that signal occur on a high to low transition.

OBSERVATION 1.1

The asterisk is inappropriate for the asynchronously running clock line SYSCLK. There is no fixed phase relationship between this clock line and other VMEbus timing.

1.5.3 Keyword numbering

This standard is based on previous standards. To preserve keyword numbering, new keywords (rules, recommendation, suggestions, permissions, observations) have been numbered starting from the last available number. Hence, keywords as presented throughout this standard are not necessary in sequential order. See keyword cross-reference index to determine where a specific keyword is referenced.

1.6 Protocol specification

There are two layers of VMEbus protocol. The lowest VMEbus layer, called the backplane access layer, is composed of the backplane interface logic, the Utility Bus modules, and the Arbitration Bus modules. The VMEbus data transfer layer, is composed of the Data Transfer Bus 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 board 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 document to specify their turn-on and turn-off times. It is shown in Figure 3.

There are two basic kinds of protocol used on the VMEbus: closed loop protocols and open loop protocols. Closed loop protocols use interlocked bus signals while open loop protocols use broadcast bus signals.

1.6.1 Interlocked bus signals

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 which is answered later with an interrupt acknowledge signal (no time limit is prescribed by the VMEbus specification). The Interrupter doesn't remove the interrupt request until the Interrupt Handler acknowledges it.

Interlocked bus signals coordinate internal functions of the VMEbus system, as opposed to interacting with external stimuli. Each interlocked signal has a source module and a destination module within the VMEbus 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 addresses and data which are the basis for all information flow between modules in the data transfer layer.

1.6.2 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, long enough to assure 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 a.c. failure lines. These signal lines are not sent to any specific module, but announce special conditions to all modules.

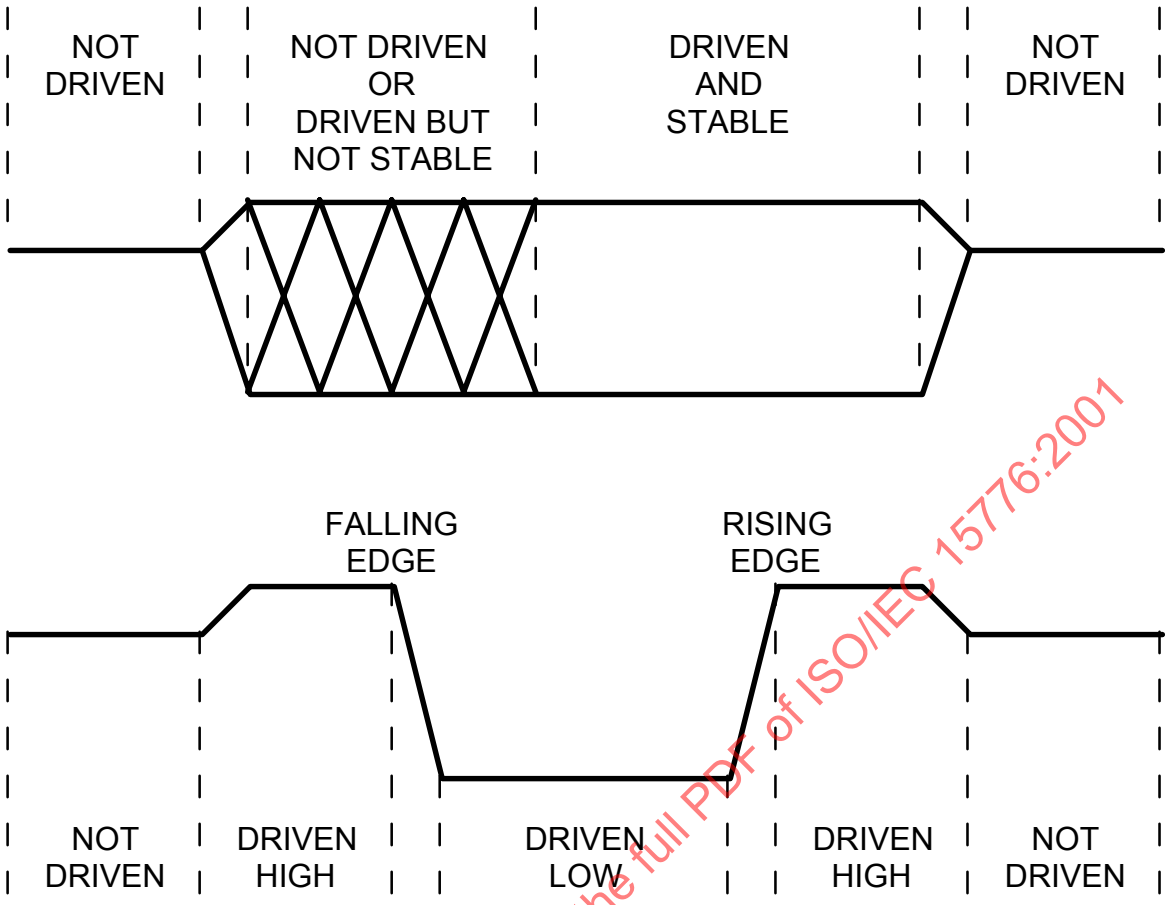


Figure 3 – Signal timing notation

1.7 System examples and explanations

A protocol specification describes, in detail, the behavior of the various functional modules. It discusses how a module responds to a signal without saying where the signal came from. Because of this, a protocol specification does not give the reader a complete picture of what is going on over the bus. To help the reader, the VMEbus specification provides examples of typical VMEbus operations. Each example shows one possible sequence of events; other sequences are also possible. In providing these examples, there is the danger that readers will assume that the sequence shown in the example is the only legal one. To help readers avoid this trap, all examples are given in a narrative style, using the present tense. This is in contrast to the imperative style used when giving rules for compliance with the VMEbus specification.

2 Data transfer bus

2.1 Introduction

This standard describes an asynchronous parallel Data Transfer Bus (DTB). Figure 4 shows a typical VMEbus system, including all of the DTB functional module types. 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.

Location Monitors monitor data transfers between Masters and Slaves. When an access is done to one of the byte location(s) 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, if processor board A writes into a location of the global VMEbus memory that is monitored by processor B's 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 VMEbus allows a Slave to take as long as it needs to respond. If a Slave fails to respond because of some malfunction, or if the Master accidentally addresses a location where there is no Slave, the Bus Timer intervenes, allowing the cycle to be terminated.

2.1.1 Enhancements

This standard is based on VMEbus Rev C.1 and includes all the capabilities described in that standard. In addition the following enhancements have been added. For faster and wider transfers, the multiplexed block transfer (MBLT) feature is included in this standard. MBLT provides 64-bit data transfers by using both the data and address lines. MBLT protocols use the same basic asynchronous protocol as in D08(EO), D16 and D32 transfers.

Additionally, 64-bit addressing modes have been added to the existing A16, A24 and A32 modes. As memory densities increase, a single VME board could use the complete A32 address range. A64 also provides support for RISC and CISC processors with full A64 addressing.

For enhanced support of P1/J1 only systems (e.g. 3U VMEbus cards), a 40-bit address mode has been included. This mode provides access to a one terabyte address space using only the P1/J1 connector. A multiplexed data cycle (MD32) has been added to allow 32-bit transfer on P1 only. MD32 uses the data bus and the address bus for 32 bit transfers. A block transfer mode (A40BLT) allows 8-, 16- and 32-bit transfers. These modes are grouped naturally with the A24 and A16 address modes. Although intended for P1/J1 only systems nothing precludes its use in full P1/P2 systems (6U VMEbus cards).

Five lock commands were added to provide multiport resource locking. This feature better supports the RISC and CISC microprocessors with a lock line output capability. A series of operations can now be performed on a resource without interference from another processor.

The Configuration ROM/Control & Status Registers (CR/CSR) provides processor, software and operating system independence for initialization, test and configuration of a board plugged into a VME backplane. The board's manufacturer and identification are also contained in the ROM area. A board's Control and Status Registers (CSR) can be implemented in the most expedient method deemed necessary by the board designer. Use of jumpers and other manual configuration are no longer needed.

VME cycles are defined to have either one phase or two phases. Single phase cycles are typically writes where both address and data are presented on the bus at the same time. Read cycles are divided into two phases, where the address is first broadcast to all Slaves followed by a data phase where the selected Slave presents data to the bus for the Master to read. For D08(E), D08(O), D16, D32 and MD32 reads there is an address broadcast phase followed by a data phase with a single data transfer. In BLT (except the first write cycle), MBLT and A40BLT cycles there is an address broadcast phase followed by a data phase with multiple data transfers. All A40 and A64 transactions also have two phases. Except for address only cycles (without handshake), all VME transfers end with one or more combinations of DTACK*, RETRY*, and BERR* being asserted low.

NOTE The Rules 2.61 through 2.65 and 2.67 are not used in this document to avoid confusion with other VMEbus specifications.

2.2 Data transfer bus lines

The Data Transfer Bus lines are grouped into three categories:

Addressing Lines	Data Lines	Control Lines
A[31..1]	D[31..0]	AS*
D[31..0]	A[31..1]	WRITE*
LWORD*	LWORD*	BERR*
AM[5..0]		DTACK*
DS0*		DS0*
DS1*		DS1*
		RETRY*

OBSERVATION 2.1

The two data strobes (DS0* and DS1*) serve a dual function.

a) The levels of these two data strobe lines are used to select which byte(s) are accessed.

b) The edges of the data strobes are also used as timing signals which coordinate the transfer of data between the Master and Slave.

OBSERVATION 2.86

A[31..1] and LWORD* serve a dual function.

a) During the address broadcast phase of all cycles, all or some of these signals are used to carry addressing information.

b) During the data transfer phase of 64-bit block transfers, all of these signals are used to carry data.

c) During the data transfer phase of A40 and A40BLT cycles, some of these signals may be used to carry data.

OBSERVATION 2.87

The data lines D[31..0] serve a dual function.

a) During the data transfer of all cycles, all or some of these signals are used to carry data.

b) During the address broadcast phase of cycles where A40 and A64 address modes are used, all or some of these signals are used to carry addressing information.

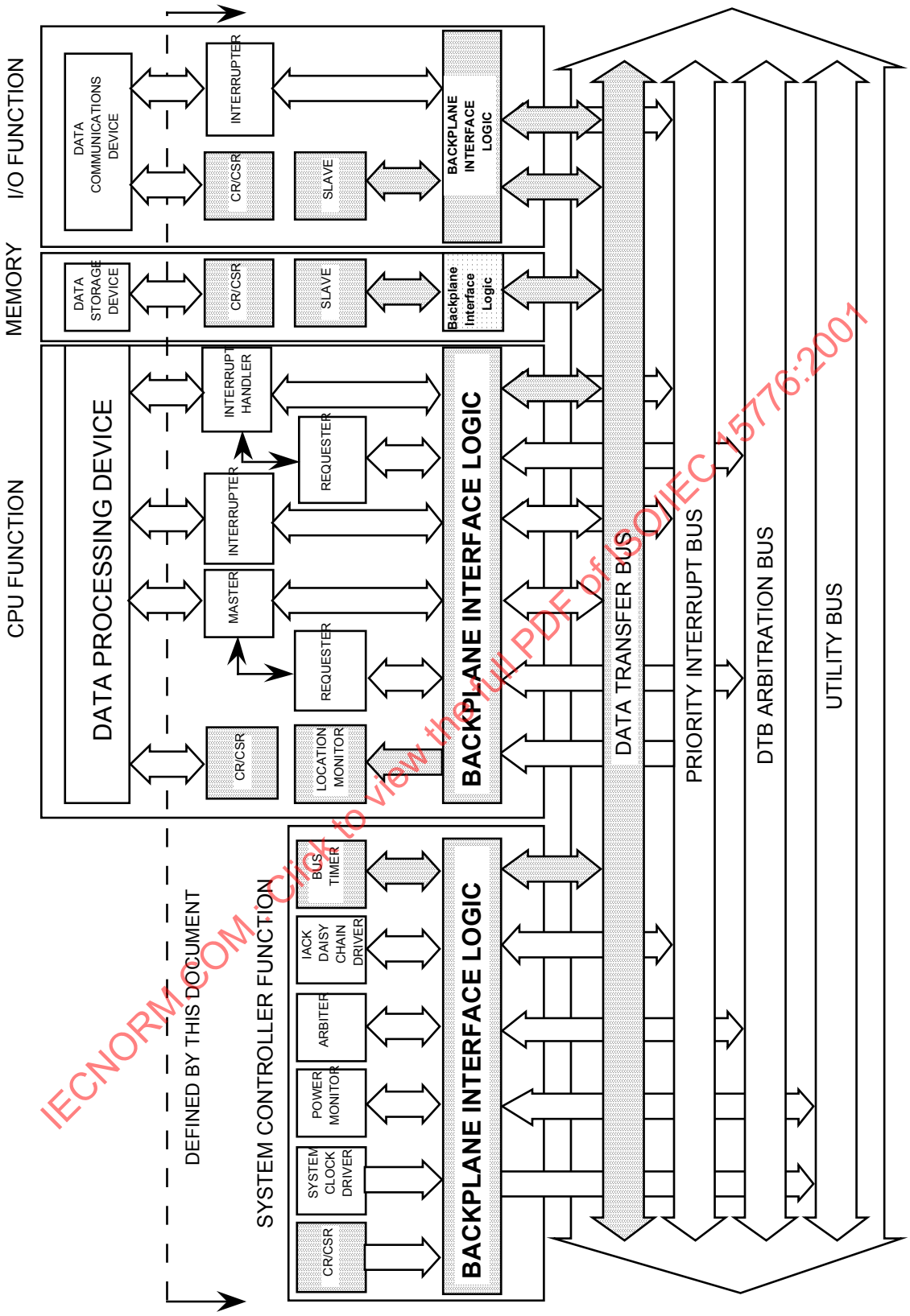


Figure 4 – Data transfer bus functional block diagram

2.2.1 Addressing lines

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

Table 1 – The eight categories of byte locations

Category in a 4-byte group	Category in an 8-byte group	Byte address
Byte(0)	Byte(0)	XXXXXX.....XXXXXX000
Byte(1)	Byte(1)	XXXXXX.....XXXXXX001
Byte(2)	Byte(2)	XXXXXX.....XXXXXX010
Byte(3)	Byte(3)	XXXXXX.....XXXXXX011
Byte(0)	Byte(4)	XXXXXX.....XXXXXX100
Byte(1)	Byte(5)	XXXXXX.....XXXXXX101
Byte(2)	Byte(6)	XXXXXX.....XXXXXX110
Byte(3)	Byte(7)	XXXXXX.....XXXXXX111

NOTE 1 Byte 0 is selected by DS1* and Byte 1 is selected by DS0*.

NOTE 2 Note that in A40 mode, Byte(1) of the 40-bit address uses D[15..08]. This reduces the cost of implementation of mixed A64 and A40 bus capabilities.

A set of byte locations whose address differs only in the two least significant bits, is referred to as a 4-byte group or a Byte(0-3) group. Some or all of the bytes in a 4-byte group can be accessed simultaneously during DTB cycles.

A set of byte locations whose address differs only in the three least significant bits, is referred to as an 8-byte group or a Byte(0-7) group. All of the bytes in an 8-byte group are always accessed simultaneously. The 8-byte group is signaled by unique AM codes.

Assignment of the address bytes and bits for A16, A24, A32, A40 and A64 is shown in Table 2.

Table 2 – Address alignment on bus

Address mode	D[31.24]	D[23..16]	D[15..8]	D[7..0]	A[31..24]	A[23..16]	A[15..8]	A[7..1]
A16							A[15..8]	A[7..1]
A24						A[23..16]	A[15..8]	A[7..1]
A32					A[31..24]	A[23..16]	A[15..8]	A[7..1]
A40			A[31..24]	A[39..32]		A[23..16]	A[15..8]	A[7..1]
A64	A[63..56]	A[55..48]	A[47..40]	A[39..32]	A[31..24]	A[23..16]	A[15..8]	A[7..1]

When accessing a 4-byte group, Masters use address lines A[63..2] or a subset of these lines to select which 4-byte group is accessed. Four additional lines, DS1*, DS0*, A1 and LWORD*, are used during the address broadcast phase to select which byte locations within the 4-byte group will be 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 3.

OBSERVATION 2.2

In cycles where both data strobes are driven low, one data strobe might go low slightly after the other. In this case the signal levels indicated in Table 3 are the final levels.

OBSERVATION 2.3

Given the 4 signal line levels shown in Table 3, there are 16 possible combinations of levels. Of these 16, there are two illegal combinations that are not used (see Table in RULE 2.1a).

RULE 2.1a

Masters **MUST** ensure that none of the following combinations are generated during the address phase:

DS1*	DS0*	A1	LWORD*
high	low	high	low
low	high	high	low

PERMISSION 2.1

Masters that generate Byte(1-2) Read or Byte(1-2) Write cycles MAY generate either of the two combinations described in RULE 2.1a briefly as transition states (i.e. while one data strobe has fallen but the other has not).

OBSERVATION 2.4

Whenever a Master drives LWORD* low and A1 high it drives both data strobes low. (Any other combination is illegal.) VMEbus board designers can take advantage of this to simplify the logic on Slaves.

PERMISSION 2.2

To simplify the required logic, Slaves which respond to Byte(1-2) Read and Byte(1-2) Write cycles MAY be designed without logic to distinguish between these cycles and the two illegal cycles described in RULE 2.1a.

As will be explained in 2.2.2, Masters use special address modifier codes to indicate that they wish to access an 8-byte group. Address lines A[63..3] or a subset of these lines are used during the address broadcast phase to select which 8-byte group will be accessed during the data transfer. DS0*, DS1*, A1 and LWORD* lines are not used to select byte-locations when selecting an 8-byte group. During the MBLT data transfer phases, LWORD* and A[31..1] are used to carry data.

Also special address modifier codes are used to indicate A40 transfer modes. During A40 modes, A[15..1] and LWORD* are used along with D[15..0] to transfer a 4-byte group.

RULE 2.69

Whenever a Master identifies a cycle as an access to an 8-byte group by using one of the 64-bit block transfer AM codes (see 2.2.2), it **MUST** drive DS0*, DS1*, LWORD*, A1 and A2 low during the address phase.

Table 3 – Signal levels during data transfers used to select which byte location(s) are accessed during a data transfer

Type of cycle	DS1*	DS0*	A1	LWORD*	A2
Address-Only with No Handshake	high	high	Note 1	Note 1	high or low
Single even byte transfers					
Byte(0) Read or Write	low	high	low	high	high or low
Byte(2) Read or Write	low	high	high	high	high or low
Single odd byte transfers					
Byte(1) Read or Write	high	low	low	high	high or low
Byte(3) Read or Write	high	low	high	high	high or low
Double byte transfers					
Byte(0-1) Read or Write	low	low	low	high	high or low
Byte(2-3) Read or Write	low	low	high	high	high or low
Quad byte transfers					
Byte(0-3) Read or Write	low	low	low	low	high or low
Single byte block transfers					
Single Byte Block Read or Write	|<--------	Note 2	------>|	high	high or low
Double byte block transfers					
Double Byte Block Read or Write	low	low	Note 3	high	high or low
Quad byte block transfers					
Quad Byte Block Read or Write	low	low	low	low	high or low
Single byte RMW transfers					
Byte(0) Read-Modify-Write	low	high	low	high	high or low
Byte(1) Read-Modify-Write	high	low	low	high	high or low
Byte(2) Read-Modify-Write	low	high	high	high	high or low
Byte(3) Read-Modify-Write	high	low	high	high	high or low
Double byte RMW transfers					
Byte(0-1) Read-Modify-Write	low	low	low	high	high or low
Byte(2-3) Read-Modify-Write	low	low	high	high	high or low
Quad byte RMW transfers					
Byte(0-3) Read-Modify-Write	low	low	low	low	high or low
Unaligned transfers					
Byte(0-2) Read or Write	low	high	low	low	high or low
Byte(1-3) Read or Write	high	low	low	low	high or low
Byte(1-2) Read or Write	low	low	high	low	high or low
Eight Byte Access					
Byte(0-7)	low	low	low	low	low

NOTE 1 During Address-Only cycles, both data strobes are maintained high, but the A1 and LWORD* lines might be either high or low.

NOTE 2 During single byte block transfers, the two data strobes are alternately driven low. Either data strobe might be driven low on the first transfer. If the first accessed byte location is Byte(0) or Byte(2), then DS1* is driven low first. If the first accessed byte location is Byte(1) or Byte(3), then DS0* is driven low first. A1 is valid only on the first data transfer (i.e. until the Slave drives DTACK* or BERR* low the first time) and might be either high or low depending upon which byte the single byte block transfer begins with. If the first byte location is Byte(0) or Byte(1), then A1 is low. If the first byte location is Byte(2) or Byte(3), then A1 is high.

NOTE 3 During a double byte block transfer, the data strobes are driven low on each data transfer. A1 is valid only on the first data transfer (i.e. until the Slave drives DTACK* or BERR* low the first time) and might be either high or low depending upon what double byte group the double byte block transfer begins with. If the first double byte group is Byte(0-1), then A1 is low. If the first double byte group is Byte(2-3), then A1 is high.

Table 3 (*continued*)

An example of a single byte block transfer cycle which starts with Byte(2) is given below:

		DS1*	DS0*	A1	LWORD*
First data transfer	Byte (2)	low	high	high	high
	Byte (3)	high	low	X	X
	Byte (0)	low	high	X	X
	Byte (1)	high	low	X	X
Last data transfer	Byte (2)	low	high	X	X
X = high or low					

An example of a double byte block transfer cycle which starts with Byte(2-3) is given below:

		DS1*	DS0*	A1	LWORD*
First data transfer	Byte (2-2)	low	low	high	high
	Byte (0-2)	low	low	X	X
	Byte (2-3)	low	low	X	X
Last data transfer	Byte (0-1)	low	low	X	X
X = high or low					

2.2.2 Address modifier lines

There are 6 address-modifier lines. The Master uses these lines to pass additional binary information to the Slave during data transfers. Table 4 lists all of the 64 possible address modifier (AM) codes and classifies each into one of three categories:

Defined
Reserved
User defined

The defined address modifier codes can be further classified into five categories as follows.

a) A16 addressing AM codes, which indicate that address lines A[15..2] are being used to select a Byte(0-3) group.

b) A24 addressing AM codes, which indicate that address lines A[23..2] are being used to select a Byte(0-3) group or address lines A[23..3] for selecting a Byte(0-7) group.

c) A32 addressing AM codes, which indicate that address lines A[31..2] are being used to select a Byte(0-3) group or address lines A[31..3] for selecting a Byte(0-7) group.

d) A40 addressing AM codes, which indicate that address lines A[23..2] and data lines D[15..0] are being used to select a Byte(0-3) group.

e) A64 addressing AM codes, which indicate that address lines A[31..2] and data lines D[31..0] are being used to select a Byte(0-3) group or address lines A[31..3] and data lines D[31..0] for selecting a Byte(0-7) group.

RULE 2.2

Except for the user-defined codes, the codes defined in Table 4 **MUST NOT** be used for purposes other than those specified.

RULE 2.3

VMEbus Slaves boards **MUST NOT** respond to reserved address-modifier codes.

OBSERVATION 2.5

Reserved address-modifier codes are for future enhancements. If Slave boards respond to these codes, incompatibilities might result at some future date, when the use 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 4 – Address modifier codes

HEX CODE	ADDRESS MODIFIER						FUNCTION
	5	4	3	2	1	0	
3F	H	H	H	H	H	H	A24 supervisory block transfer (BLT)
3E	H	H	H	H	H	L	A24 supervisory program access
3D	H	H	H	H	L	H	A24 supervisory data access
3C	H	H	H	H	L	L	A24 supervisory 64-bit block transfer (MBLT)
3B	H	H	H	L	H	H	A24 non-privileged block transfer (BLT)
3A	H	H	H	L	H	L	A24 non-privileged program access
39	H	H	H	L	L	H	A24 non-privileged data access
38	H	H	H	L	L	L	A24 non-privileged 64-bit block transfer (MBLT)
37	H	H	L	H	H	H	A40BLT
36	H	H	L	H	H	L	Reserved
35	H	H	L	H	L	H	A40 lock command (LCK)
34	H	H	L	H	L	L	A40 access
33	H	H	L	L	H	H	Reserved
32	H	H	L	L	H	L	A24 lock command (LCK)
31	H	H	L	L	L	H	Reserved
30	H	H	L	L	L	L	Reserved
2F	H	L	H	H	H	H	Configuration ROM/Control & Status Register (CR/CSR)
2E	H	L	H	H	H	L	Reserved
2D	H	L	H	H	L	H	A16 supervisory access
2C	H	L	H	H	L	L	A16 lock command (LCK)
2B	H	L	H	L	H	H	Reserved
2A	H	L	H	L	H	L	Reserved
29	H	L	H	L	L	H	A16 non-privileged 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
21	H	L	L	L	L	H	Reserved
20	H	L	L	L	L	L	Reserved
L = low-signal level							H = high-signal level

Table 4 *(continued)*

HEX CODE	ADDRESS MODIFIER						FUNCTION
	5	4	3	2	1	0	
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	A32 supervisory block transfer (BLT)
0E	L	L	H	H	H	L	A32 supervisory program access
0D	L	L	H	H	L	H	A32 supervisory data access
0C	L	L	H	H	L	L	A32 supervisory 64-bit block transfer (MBLT)
0B	L	L	H	L	H	H	A32 non-privileged block transfer (BLT)
0A	L	L	H	L	H	L	A32 non-privileged program access
09	L	L	H	L	L	H	A32 non-privileged data access
08	L	L	H	L	L	L	A32 non privileged 64-bit block transfer (MBLT)
07	L	L	L	H	H	H	Reserved
06	L	L	L	H	H	L	Reserved
05	L	L	L	H	L	H	A32 lock command (LCK)
04	L	L	L	H	L	L	A64 lock command (LCK)
03	L	L	L	L	H	H	A64 block transfer (BLT)
02	L	L	L	L	H	L	Reserved
01	L	L	L	L	L	H	A64 single transfer access
00	L	L	L	L	L	L	A64 64-bit block transfer (MBLT)
L = low-signal level							H = high-signal level

RECOMMENDATION 2.1

VMEbus users MAY 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 for their system.

2.2.3 Data lines

VMEbus systems can be built with a backplane configuration that provides either 16 data lines, D[15..0], or 32 data lines, D[31..0]. Backplane configurations that provide 16 data lines allow a Master to access up to 2 or 4 byte locations simultaneously, while those with 32 data lines allow it to access up to 8 byte locations simultaneously. When the Master has selected 1, 2, 3, 4 or 8 byte locations, using the method described in 2.2.1 and 2.2.2, it can transfer binary data between itself and those locations. Table 5 shows how the data lines and the address lines are used to access byte locations.

RULE 2.70

When accessing Byte(0-7), LWORD* **MUST** carry the least significant bit of Byte(3) and A7 **MUST** carry the most significant bit of Byte(3). When accessing Byte(0-3) in MD32 Mode, LWORD* **MUST** carry the least significant bit of Byte(1) and A7 **MUST** carry the most significant bit of Byte(1). (See Table 6.)

PERMISSION 2.4

The data sender (Master for a write cycle; Slave for a read cycle) MAY drive data lines which are not used to transfer data.

Table 5 – Use of data lines to move data during nonmultiplexed data transfers

During the following types of cycles...	the data lines are used to transfer data as shown below:			
	D[31..24]	D[23..16]	D[15..8]	D[7..0]
Address-Only	<---------- no bytes transferred ---------->			
Single even byte transfers Byte(0) Read, Write or RMW Byte(2) Read, Write or RMW			Byte(0) Byte(2)	
Single odd byte transfers Byte(1) Read, Write or RMW Byte(3) Read, Write or RMW				Byte(1) Byte(3)
Double byte transfers Byte(0-1) Read, Write or RMW Byte(2-3) Read, Write or RMW			Byte(0) Byte(2)	Byte(1) Byte(3)
Quad byte transfers Byte(0-3) Read, Write or RMW	Byte(0)	Byte(1)	Byte(2)	Byte(3)
Single byte block transfers Single Byte Block Read or Write			<-------- Note 1 -------->	
Double byte block transfers Double Byte Block Read or Write			<--------- Note 2 -------->	
Quad byte block transfers Quad Byte Block Read or Write	Byte(0)	Byte(1)	Byte(2)	Byte(3)
Unaligned transfers Byte(0-2) Read or Write Byte(1-3) Read or Write Byte(1-2) Read or Write	Byte(0)	Byte(1) Byte(1) Byte(1)	Byte(2) Byte(2) Byte(2)	Byte(3)
*NOTE RMW stands for Read-Modify-Write.				

Table 5 *(continued)*

Examples

a) During single byte block transfers, data is transferred 8 bits at a time over D[15..8] or D[7..0]. One example of this is given below:

	D[31..24]	D[23..16]	D[15..8]	D[7..0]
First data transfer				Byte(1)
			Byte(2)	
				Byte(3)
			Byte(0)	
				Byte(1)
			Byte(2)	
Last data transfer				Byte(3)

b) During double byte block transfers, data is transferred 16 bits at a time over D[15..0]. One example of this is given below:

	D[31..24]	D[23..16]	D[15..8]	D[7..0]
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 6 – Use of the address and data lines for multiplexed data cycles

	A[31..24]	A[23..16]	A[15..8]	A[7..1] LWORD*	D[31..24]	D[23..16]	D[15..8]	D[7..0]
Multiplexed Quad Byte transfers (MD32) Byte (0-3) Read, Write or RMW			Byte(0)	Byte(1)			Byte(2)	Byte(3)
Multiplexed Quad Byte Block transfers (MD32) Byte (0-3) Read or Write			Byte(0)	Byte(1)			Byte(2)	Byte(3)
Multiplexed Eight Byte Block Transfer (MBLT) Byte (0-7) Read or Write	Byte(0)	Byte(1)	Byte(2)	Byte(3)	Byte(4)	Byte(5)	Byte(6)	Byte(7)

2.2.4 Data transfer bus control lines

The following signal lines are used to control the transfer 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
RETRY*	Retry
WRITE*	Read/Write

2.2.4.1 AS*

A falling edge on AS* informs all Slaves that the addressing information on all or some of the A[31..1] lines, LWORD* and AM(5..0) are 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 valid data from the data bus.

When using the 64-bit or 40-bit addressing mode, the first falling edge of a data strobe indicates that the Master has placed valid addressing information on the data lines.

OBSERVATION 2.7

VMEbus 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 2.5a

Slaves and Location Monitors that do not support block transfer capability as described in 2.3.7 MAY be designed to capture AM[5..0], A[31..1] and LWORD* when they detect a falling edge on a data strobe instead of on the falling edge of AS*.

OBSERVATION 2.8

VMEbus Slaves and Location Monitors that capture the address on the falling edge of the data strobe(s) need not monitor AS*.

OBSERVATION 2.9a

To perform block read and write cycles as described in 2.3.7 or to take full advantage of address pipelining as described in 2.4.2, a Slave should capture the address on the falling edge of the data strobe (DS0* or DS1*) that falls first while AS* is low.

2.2.4.3 DTACK*

On a Write cycle, the Slave drives DTACK* low to indicate that it has successfully received all of the data called for by the transfer type. On a read cycle, the Slave drives DTACK* low to indicate that it has placed data on the data bus. During A64, A40, MBLT, and A40BLT address phases, DTACK* is driven low by the Slave to indicate it can receive or send data in the data phase. For lock commands the Slave drives DTACK* low indicating it has accepted the lock command and has locked out the other ports.

2.2.4.3.1 Rescinding DTACK*

Slaves are allowed to drive DTACK* high before releasing it at the end of a cycle, or during BLT, MBLT and A40BLT operations. Driving DTACK* high, versus letting DTACK* be pulled high by the backplane termination resistors provides less delay at the end of the cycle or data transfer and improves system performance. Figure 36 shows the timing parameters for Rescinding DTACK*. The timing parameter values are listed in Table 27.

PERMISSION 2.14

A selected Slave MAY drive DTACK* high for a short time, as specified by Timing Parameter 30A at the end of a cycle and during BLT, MBLT and MD32 data transfer phases after the data strobes go high.

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 or address broadcast was unsuccessful or an error was detected. For example, when a Master tries to write to a location that contains Read-Only Memory, the responding Slave should 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 time-out period has expired.

RECOMMENDATION 2.2

Design Slaves to drive BERR* low when they detect an uncorrectable error in the data they retrieve from their internal storage during a read cycle. Also, design Slaves to drive BERR* low when a bus operation tries to write to devices that are read only.

PERMISSION 2.6

VMEbus Slaves MAY be designed without a BERR* driver.

2.2.4.5 WRITE*

WRITE* is a level-significant signal line. 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.

2.2.4.6 RETRY*

RETRY* can be driven low by the responding Slave to indicate to the Master that the requested data transfer cannot be executed at this time. The Master that supports RETRY* should retry the data transfer at a later time. This operation may be used to break deadlock situations.

The Master waits for a short period of time before trying the operation again. In the interim, the bus can be used for other transactions.

See 2.3.13 for rules, recommendations, permissions and observations governing the use of the RETRY* line.

2.3 DTB modules – Basic description

The DTB protocols define the methods used to transfer data. Five addressing modes are provided:

a) A16 (16-bit)
b) A24 (24-bit)
c) A32 (32-bit)
d) A40 (40-bit)
e) A64 (64-bit).

The capabilities of Masters, Slaves, Location Monitors and CR/CSRs are described by a list of mnemonics that show what cycle types they can generate, accept or monitor respectively. (This will be described in more detail later in this chapter.)

Subclauses 2.3.1 through 2.3.5 provide block diagrams for the five types of DTB functional modules: Master, Slave, Location Monitor, Bus Timer and CR/CSR.

RULE 2.7

Output signal lines shown with solid lines in Figures 5 through 8 **MUST** be driven by the respective module, unless it would always drive them high.

OBSERVATION 2.11

IF 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 Figures 5 through 8 **MUST** 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 Figures 5 through 8 are given in Tables 8, 9, and 11.

2.3.1 Master

The block diagram of the Master is shown in Figure 5. The dotted lines in the diagram show signals whose use varies among the various types of Masters. Table 7 specifies how the various types of Masters use these lines. Further information about how the various types of Masters drive the address line, the data lines, LWORD*, DS0*, DS1* and A1 is given in Tables 3, 24, 25.

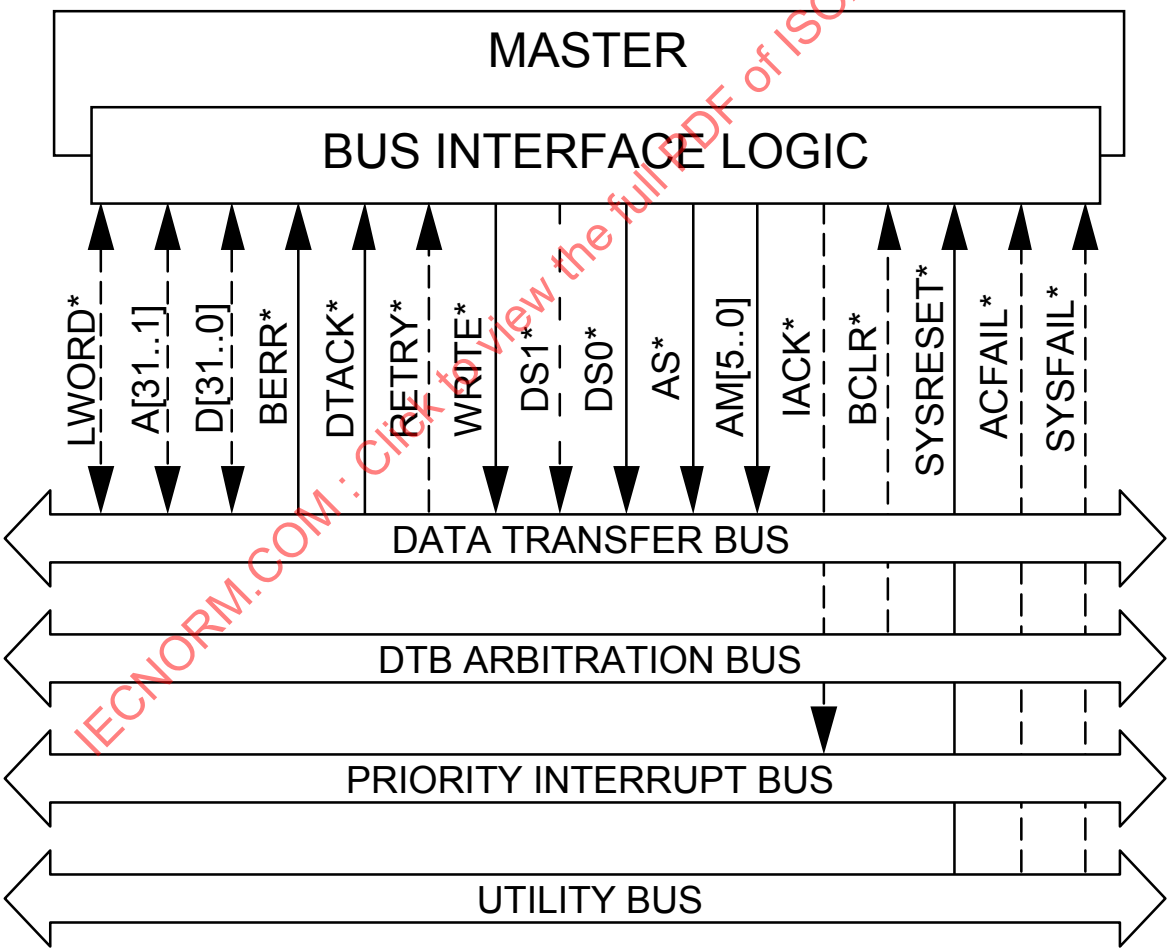


Figure 5 – Block diagram – Master

Table 7 – RULES and PERMISSIONS specifying the use of the dotted lines by the various types of masters

Type of master	Use of dotted lines
D08(O)	**MUST** monitor and drive D[7..0] **MAY** monitor and drive LWORD* **MAY** monitor and drive D[31..8]
D08(EO)	**MUST** drive DS0* or DS1, but not both low on the same data transfer **MUST** monitor and drive D[15..0] **MAY** monitor and drive LWORD* **MAY** monitor and drive D[31..16]
D16 Single & BLT	**MUST** drive DS1* **MUST** monitor and drive D[15..0] **MAY** monitor and drive LWORD* **MAY** monitor and drive D[31..16]
D32 Single & BLT	**MUST** drive DS1* **MUST** drive LWORD* **MUST** monitor and drive D[31..0] **MAY** monitor LWORD*
MD32	**MUST** drive DS1* **MUST** monitor and drive LWORD* **MUST** monitor and drive D[15..0] **MUST** monitor and drive A[15..1]
MBLT	**MUST** drive DS1* **MUST** monitor and drive LWORD* **MUST** monitor and drive D[31..0] **MUST** monitor and drive A[31..1]
A16	**MUST** drive A[15..1] **MAY** drive A[31..16]
A24	**MUST** drive A[23..1] **MAY** drive A[31..24]
A32	**MUST** drive A[31..1]
A40 & A40BLT	**MUST** drive DS1* **MUST** drive A[23..1] **MUST** drive D[15..0] **MUST** drive LWORD*
A64	**MUST** drive DS1* **MUST** drive A[31..1] **MUST** drive D[31..0] **MUST** drive LWORD*
ALL	**MAY** monitor RETRY* **MAY** monitor BCLR* (see clause 3) **MAY** monitor ACFAIL* (see clause 5) **MAY** monitor SYSFAIL* (see clause 5) **MUST NOT** drive IACK* low

NOTE 1 The mnemonics A16, A24, A32, A40 and A64 are defined in Table 11.

NOTE 2 The mnemonics D08(EO), D16, D32 and MD32 are defined in Table 12.

NOTE 3 The mnemonics BLT, MBLT and A40BLT are defined in Table 13.

2.3.2 Slave

The block diagram of the Slave is shown in Figure 6. The dotted lines in the diagram show signals whose use varies among the various types of Slaves. Table 8 shows how the various types of Slaves use these lines. Further information about how the various types of Slaves drive the data lines is given in Table 27.

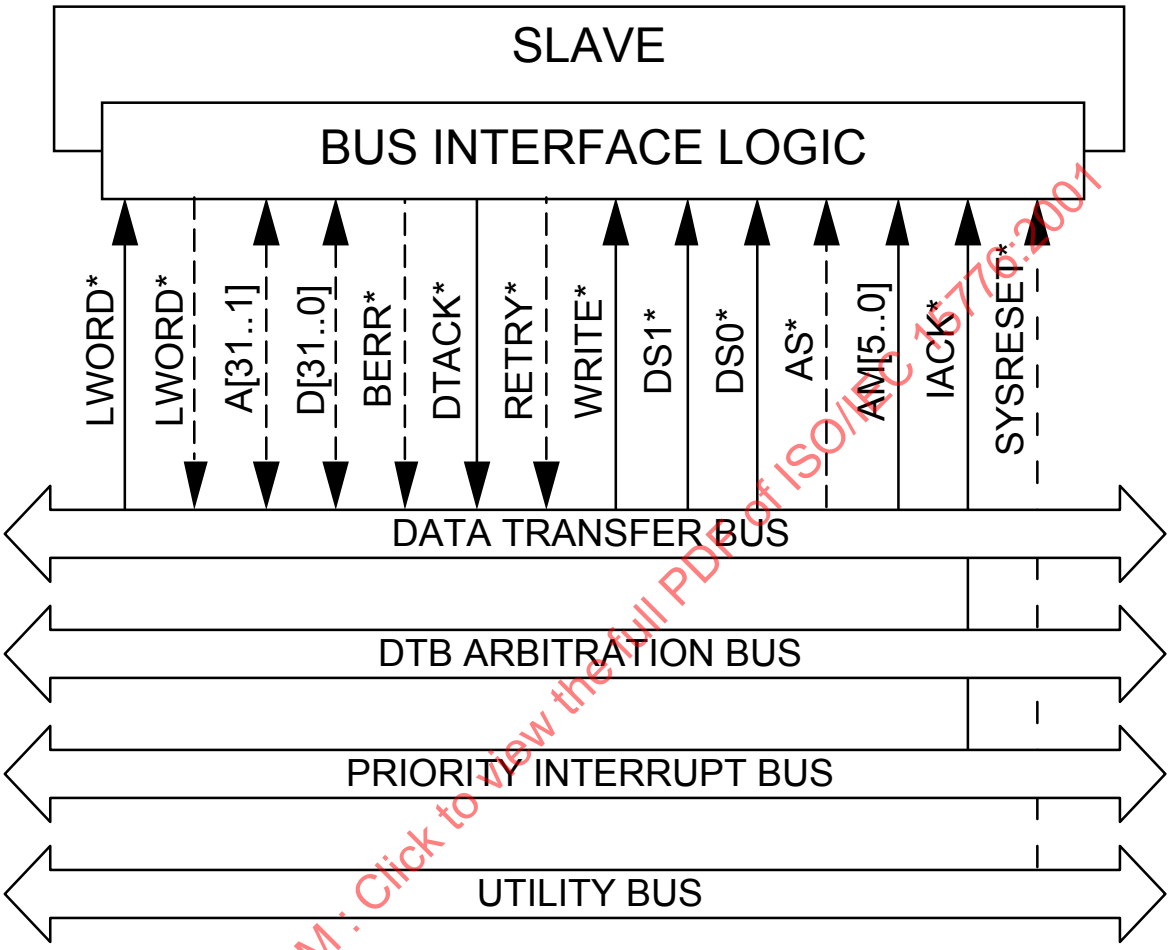


Figure 6 – Block diagram – Slave

Table 8 – Slaves – RULEs and PERMISSIONs specifying the use of the dotted lines by the various types of slaves

Type of slave	Use of dotted lines
D08(O)	**MUST** monitor and/or drive D[7..0] **MAY** monitor AS* **MAY** monitor or drive D[31..8]
D08(EO) and D16	**MUST** monitor and/or drive D[15..0] **MAY** monitor AS* **MAY** monitor or drive D[31..16]
D32	**MUST** monitor and/or drive D[31..0] **MAY** monitor AS*
MD32	**MUST** monitor and/or drive A[15..1] **MUST** monitor and/or drive LWORD* **MUST** monitor and/or drive D[15..0] **MUST** monitor AS*
MBLT	**MUST** monitor and/or drive D[31..0] **MUST** monitor and/or drive A[31..1] **MUST** monitor AS* **MUST** monitor and/or drive LWORD*
A16	**MUST** monitor A[15..1] **MAY** monitor A[31..16]
A24	**MUST** monitor A[23..1] **MAY** monitor A[31..24]
A32	**MUST** monitor A[31..1]
A40 & A40BLT	**MUST** monitor D[15..0] **MUST** monitor A[23..1] **MUST** monitor AS*
A64	**MUST** monitor A[31..1] **MUST** monitor D[31..0] **MUST** monitor AS*
ALL	**MAY** drive RETRY* **MAY** drive BERR* **MAY** monitor SYSRESET*
NOTE 1 The mnemonics D08(O), D08(EO), D16 and D32 are defined in Table 12. NOTE 2 The mnemonics BLT, MBLT and A40BLT are defined in Table 13. NOTE 3 The mnemonics A16, A24, A32, A40 and A64 are defined in Table 11.	

2.3.3 Bus timer

The block diagram of the Bus Timer is shown in Figure 7. Bus Timers can be designed to drive BERR* low after various periods of time. Table 9 shows how the BTO( ) mnemonic is used to describe the various types of Bus Timers.

OBSERVATION 2.13

The dashed DTACK* and BERR* lines shown in Figure 7 allow designers to implement a bus timer in one of two ways:

a) 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;

b) 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.

Table 9 – Use of BTO( ) mnemonic specifying 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 *x* µs, without DTACK* being asserted low

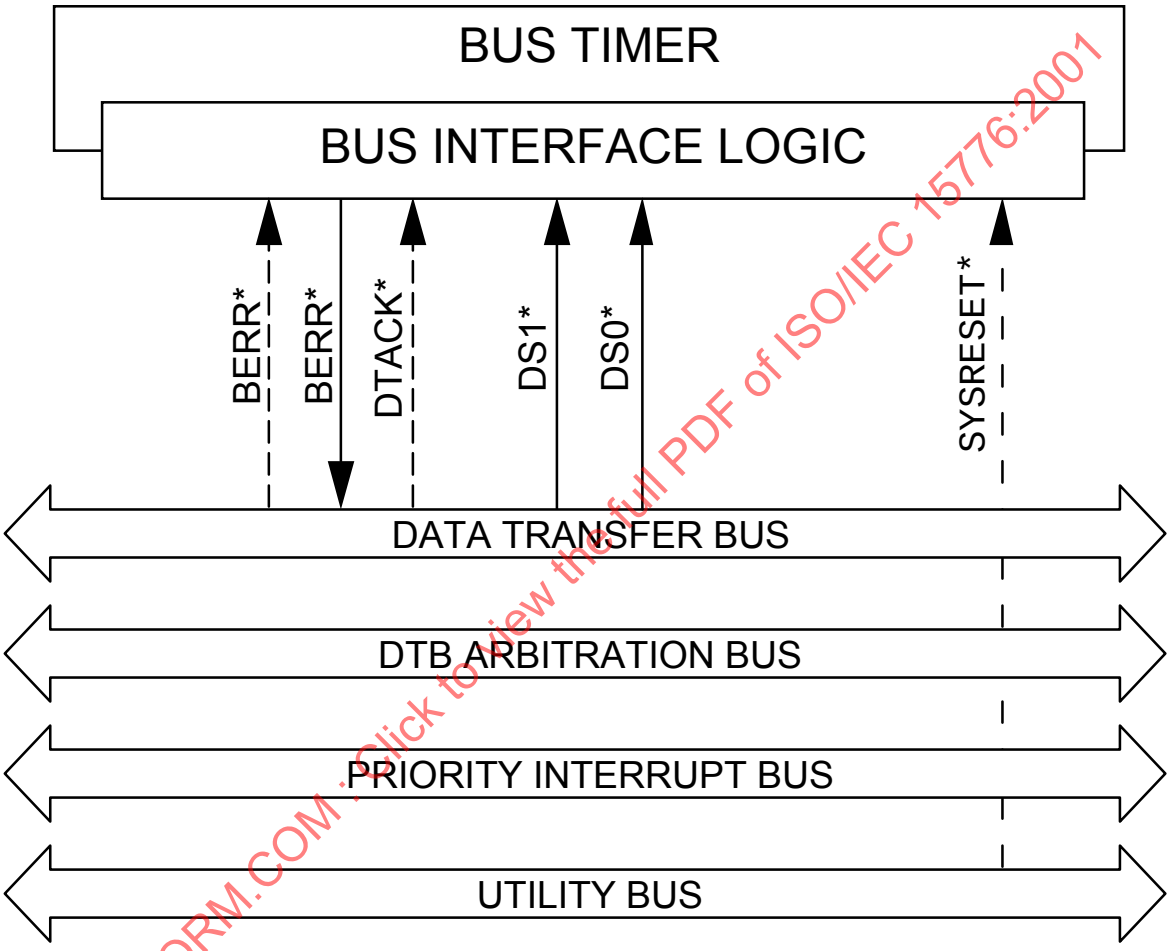


Figure 7 – Block diagram – Bus timer

2.3.4 Location monitor

The block diagram of the Location Monitor is shown in Figure 8. The dotted lines in the diagram show signals whose use varies among the various types of Location Monitors. Table 10 shows how the various types of Location Monitors use these lines.

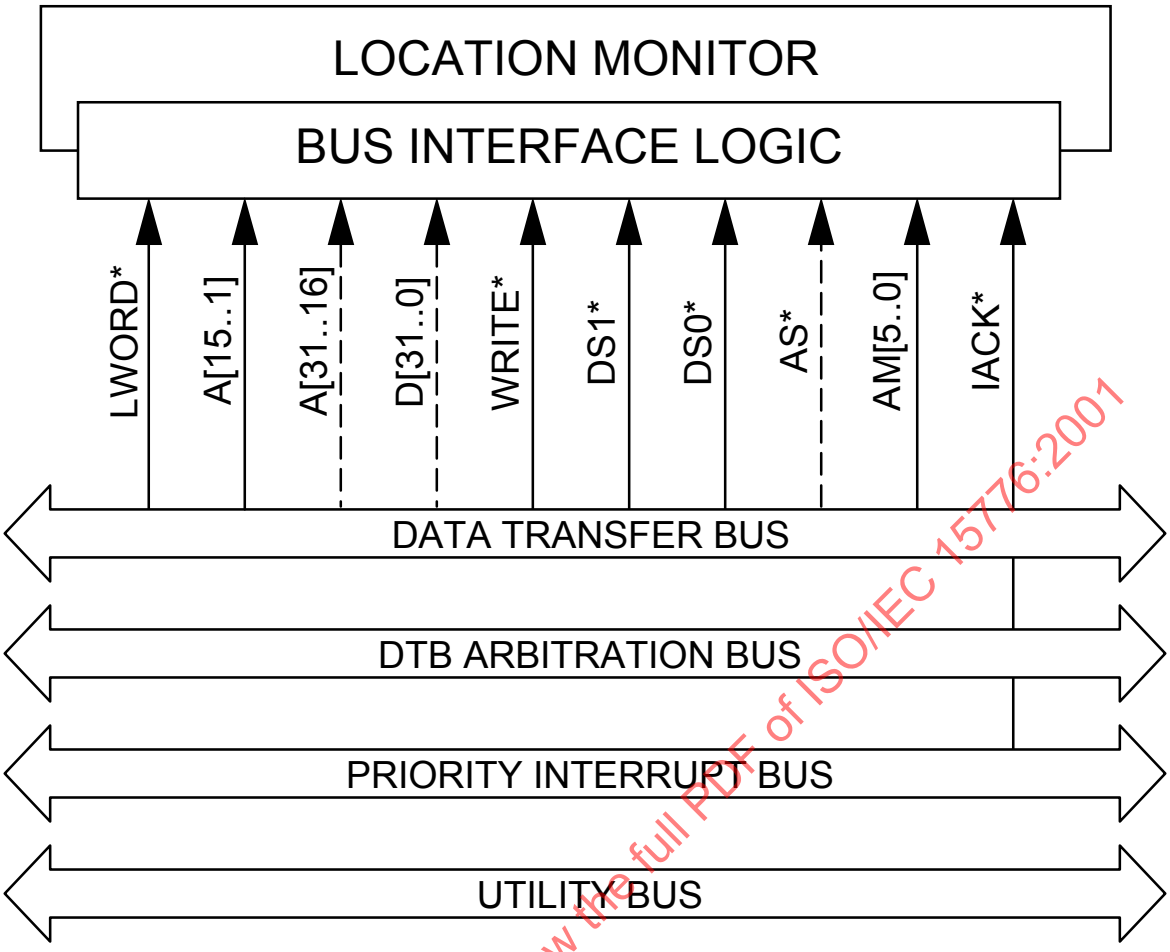


Figure 8 – Block diagram – Location monitor

Table 10 – Location monitors – RULEs and PERMISSIONs specifying the use of dotted lines by various types of location monitors

Type of slave	Use of dotted lines
A16	**MUST** monitor A[15..1] **MAY** monitor A[31..1]
A24	**MUST** monitor A[23..1] **MAY** monitor A[31..24]
A32	**MUST** monitor A[31..1]
A40	**MUST** monitor A[23..1] **MUST** monitor D[15..0] **MUST** monitor AS*
A64	**MUST** monitor A[31..1] **MUST** monitor D[31..0] **MUST** monitor AS*
ALL	**MAY** monitor AS*
NOTE The mnemonics A16, A24, A32, A40 and A64 are defined in Table 11.	

2.3.5 Addressing phases and modes

Masters broadcast an address over the DTB at the beginning of each cycle. This broadcast address might be a 16-bit, a 24-bit, a 32-bit, a 40-bit or a 64-bit address, depending on the capabilities of the Master broadcasting it. 16-bit addresses are referred to as A16 addresses, 24-bit addresses are referred to as A24 addresses, 32-bit addresses are referred to as A32 addresses, 40-bit addresses are referred to as A40 addresses and 64-bit addresses are referred to as A64 addresses.

Table 11 shows the various mnemonics used to describe the addressing capabilities and how each is used to describe Masters, Slaves, Location Monitors and CR/CSRs.

The Master broadcasts an Address Modifier (AM) code along with each address to tell Slaves whether the address is A16, A24, A32, A40 or A64.

RULE 2.9

Slave boards **MUST** decode all of the address modifier lines.

RULE 2.71

When broadcasting a 64-bit address, A64 Masters **MUST** drive the least significant 32-bits of the address on LWORD* and A[31..1] and the most significant 32-bits of address on D[31..0]. Refer to Table 2.

OBSERVATION 2.14a

Decoding all the address modifier lines allows a Slave to differentiate A16, A24, A32, A40 or A64 address mode as well as the specific operation within each address mode.

OBSERVATION 2.15a

In addition to the five modes of addressing described here, there is a sixth mode that is used on interrupt acknowledge cycles (see clause 4). These interrupt acknowledge cycles can be distinguished from data transfer cycles by the fact that IACK* is low instead of high.

RULE 2.10

Whenever a Master broadcasts an address, it **MUST** 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 **MUST NOT** respond to DTB cycles when IACK* is low.

OBSERVATION 2.88

Systems may include a mixture of A16, A24, A32, A40 and A64 Masters and Slaves.

Table 11 – Mnemonics specifying addressing capabilities

The following mnemonic	when applied to a	means that it
A16	Master Slave Location Monitor	**MUST** generate cycles with A16 addresses **MUST** accept cycles with A16 addresses **MUST** monitor cycles with A16 addresses
A24	Master Slave Location Monitor CR/CSR	**MUST** generate cycles with A24 & A16 addresses **MUST** accept cycles with A24 addresses **MUST** monitor cycles with A24 addresses **MUST** monitor cycles with A24 addresses
A32	Master Slave Location Monitor	**MUST** generate cycles with A32, A24 & A16 addresses **MUST** accept cycles with A32 addresses **MUST** monitor cycles with A32 addresses
A40	Master Slave Location Monitor	**MUST** generate cycles with A40, A24 & A16 addresses **MUST** accept cycles with A40 addresses **MUST** monitor cycles with A40 addresses
A64	Master Slave Location Monitor	**MUST** generate cycles with A64, A32, A24 & A16 addresses **MUST** accept cycles with A64 addresses **MUST** monitor cycles with A64 addresses

RULE 2.72

A64 Masters **MUST** include the A32, A24 and A16 capabilities.

RULE 2.73

A40 Masters **MUST** include the A24 and A16 capabilities.

RULE 2.74

A32 Masters **MUST** include the A24 and A16 capabilities.

RULE 2.75

A24 Masters **MUST** include the A16 capability.

SUGGESTION 2.6a

Do not assume that the above RULEs are known to the readers of product specifications. Rather, specify an A64 Master product as a "A64, A32, A24 and A16 Master". Specify an A32 Master product as a "A32, A24 and A16 Master". Similarly, specify an A24 Master product as a "A24 and A16 Master".

OBSERVATION 2.89

A64 and A32 Masters MAY include A40 capability for interoperability with 3U products.

2.3.6 Basic data transfer capabilities

There are five basic data transfer capabilities associated with the DTB: D08(O) (Odd byte only), D08(EO) (Even and Odd byte), D16, D32 and MD32. 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(EO) 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 ICs 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 D[7..0].

RULE 2.76

D16 Slaves **MUST** include D08(EO) capability.

RECOMMENDATION 2.3

D16 Masters should include D08(EO) capability.

RULE 2.77

D32 and MD32 Slaves **MUST** include D16 and D08(EO) capabilities.

RECOMMENDATION 2.4

D32 and MD32 Masters should include D16 and D08(EO) capabilities.

RECOMMENDATION 2.5

MBLT Masters should include D32, D16 and D08(EO) capabilities.

SUGGESTION 2.9

Do not assume that the above RULEs are known to the readers of product specifications. Specify products that have the D32 capability as "D32, D16 and D08(EO)". Similarly, specify D16 products as "D16 and D08(EO)".

RULE 2.4

D08(O), D08(EO), and D16 Slaves **MUST NOT** respond with a falling edge on DTACK* during a quad byte cycle if they do not have quad-byte capability.

RULE 2.5

D08(O) and D08(EO) Slaves **MUST NOT** respond with a falling edge on DTACK* during a double-byte cycle if it does not have double-byte capability.

SUGGESTION 2.8a

Design Slaves are to respond with a falling edge on BERR* in the following situations:

a) when a D08(O), D08(EO) or D16 only Slave is requested to do a quad-byte cycle;

b) when a D08(O) or D08(EO) Slave is requested to do a double-byte cycle;

c) when a D08(O), D08(EO) or D16 Slave is requested to do an unaligned transfer (that is, a triple-byte transfer or a double-byte Byte(1-2) transfer);

d) when a DO8(O) only Slave is requested to do an even single-byte transfer (that is, a single-byte Byte(0) or Byte(2) transfer).

Table 12 shows the various mnemonics used to describe the basic data transfer capabilities and how each is used to describe Masters, Slaves and Location Monitors.

Table 12 – Mnemonics specifying basic data transfer capabilities

The following mnemonic	when applied to a	means that it
D08(EO)	Master Slave Location Monitor	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** 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
D08(O)	Slave	**MUST** accept the following cycles: Single byte read cycles: Byte(1) Read Byte(3) Read Single byte write cycles: Byte(1) Write Byte(3) Write
D16	Master Slave Location Monitor	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** 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	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Quad byte read cycle: Byte(0-3) Read Quad byte write cycle: Byte(0-3) Write
MD32	Master Slave Location Monitor	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Multiplexed Quad byte read cycle: Byte(0-3) Read Multiplexed Quad byte write cycle: Byte(0-3) Write
NOTE (EO) is Even and Odd; (O) is Odd only.		

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).

Two types of block data transfer cycles are defined: basic block transfer cycles (BLT and A40BLT) and a multiplexed block transfer (MBLT) cycle. During standard block transfer cycles the Master can simultaneously access either 1, 2 or 4 byte locations in the course of each data transfer. During multiplexed block transfers, the Master simultaneously accesses 8-byte locations (MBLT) in each data transfer.

Both the data lines and the address lines are used to transfer data during MBLT and MD32 data transfers.

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.

The number of transfers in a block is limited to 256. The intent is to limit the length of time a block transfer can occupy the bus and prevent an arbitration cycle.

OBSERVATION 2.90

As described in 2.2.2, the Master differentiates BLT, MBLT, and A40BLT cycles by driving special codes on the address modifier lines.

OBSERVATION 2.18a

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. The following RULEs were formulated to simplify the design of these Slaves and to permit faster access time.

RULE 2.12a

D08(EO), D16, D32 and MD32 block transfer cycles (BLT) **MUST NOT** cross any 256 byte boundary.

RULE 2.78

MBLT cycles **MUST NOT** cross any 2048 byte boundary.

OBSERVATION 2.19a

The above RULEs, which establish block transfer boundaries, limit the maximum length of block transfers. However, knowing that only A[7..1] or A[10..3] can change during 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 2.20a

In some cases it might be necessary to transfer a large block of data that crosses one or more block transfer boundaries. In such a case, when the Master that does the block transfer is designed to recognize the arrival at a block transfer boundary, it can drive AS* high for the required minimum period and then initiate another block transfer without the intervention of system software.

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 **MUST** monitor AS* and capture the addressing information when they detect a falling edge on AS*.

OBSERVATION 2.91

Address is stable from AS* low to DTACK* low. AS* will stay low when transitioning from an address phase to a data transfer phase.

OBSERVATION 2.92

D32 BLT Masters and Slaves and MD32 BLT Masters and Slaves are not required to support D16 BLT and D08(EO) BLT. Likewise, D16 BLT Masters and Slaves are not required to support D08(EO) BLT.

RULE 2.79

During the course of a single BLT cycle, a Master **MUST NOT** mix single-byte, double-byte and quad-byte data transfers.

Table 13 lists the mnemonics used to describe block transfer capabilities and how they are used to describe the various types of Masters, Slaves and Location Monitors.

OBSERVATION 2.93

A block transfer could span a range of addresses which may include one that is monitored by a Location Monitor. Therefore, when the mnemonics BLT and MBLT are applied to a Location Monitor, it means that the Location Monitor includes address counters that allow it to recognize when one of the locations which it was configured to monitor was accessed during a block transfer cycle.

RULE 2.80

As shown in Table 13, a Master **MUST NOT** execute a block transfer cycle that accesses byte locations using unaligned transfers (see 2.3.9).

Table 13 – Mnemonics specifying block transfer capabilities

The following mnemonic	when applied to a	means that it
BLT and A40BLT	D08(EO) Master D08(EO) Slave D08(EO) Loc Mon	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Block read cycles: Single-Byte Block Read Block write cycles: Single-Byte Block Write
BLT and A40BLT	D16 Master D16 Slave D16 Loc Mon	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Block read cycles: Double-Byte Block Read Block write cycles: Double-Byte Block Write
BLT	D32 Master D32 Slave D32 Loc Mon	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Block read cycles: Quad-Byte Block Read Block write cycles: Quad-Byte Block Write
A40BLT	MD32 Master MD32 Slave MD32 Loc Mon	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Multiplexed Quad Block read cycles: Quad-Byte Block Read Multiplexed Quad Block write cycles: Quad-Byte Block Write
MBLT	MBLT Master MBLT Slave MBLT Loc Mon	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Multiplexed Block read cycles: Eight-Byte Block Read Multiplexed Block write cycles: Eight-Byte Block Write

2.3.8 Read-Modify-Write capability

In multiprocessor systems which 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 an example:

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

This conflict is avoided by defining a Read-Modify-Write Cycle or Lock command which 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.

The same applies for a resource that can be accessed locally and globally over the bus. Whenever a resource is being accessed by the bus, with AS* low, local access is to be locked out. With this type of operation, a global processor can be assured that when it runs a Read-Modify-Write cycle on a dual-ported resource it will be run atomically.

Table 14 lists the various mnemonics used to describe read-modify-write capabilities and how each is used to describe Masters, Slaves, and Location Monitors.

Table 14 – The mnemonic that specifies Read-Modify-Write capabilities

The following mnemonic	when applied to a	means that it
RMW	D08(EO) Master D08(EO) Slave D08(EO) Location Monitor	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Single byte Read-Modify-Write cycles: Byte(0) Read-Modify-Write Byte(1) Read-Modify-Write Byte(2) Read-Modify-Write Byte(3) Read-Modify-Write
	D08(O) Slave	**MUST** accept the following cycles: Single byte Read-Modify-Write cycles: Byte(1) Read-Modify-Write Byte(3) Read-Modify-Write
	D16 Master	**MUST** generate the following cycles:
	D16 Slave	**MUST** accept the following cycles:
	D16 Location Monitor	**MUST** monitor the following cycles: Double byte Read-Modify-Write cycles: Byte(0-1) Read-Modify-Write Byte(2-3) Read-Modify-Write
	D32 Master D32 Slave D32 Location Monitor	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Quad byte Read-Modify-Write cycles: Byte(0-3) Read-Modify-Write
	MD32 Master MD32 Slave MD32 Location Monitor	**MUST** generate the following cycles: **MUST** accept the following cycles: **MUST** monitor the following cycles: Multiplexed Quad byte Read-Modify-Write cycles: Byte(0-3) Read-Modify-Write

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 Figure 9.

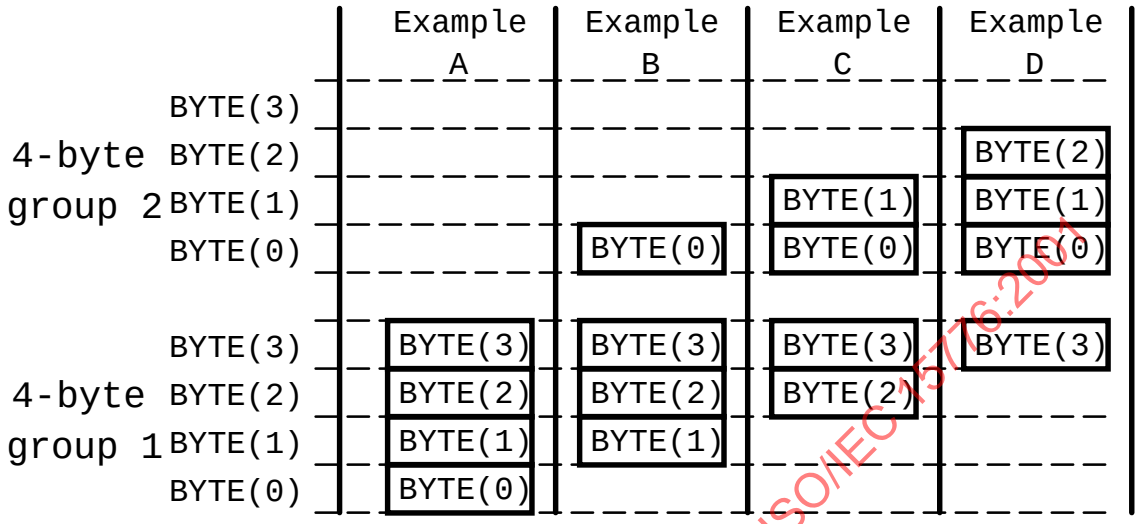


Figure 9 – Four ways in which 32 bits of data might be stored in memory

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 15.

OBSERVATION 2.22

The sequences shown in Table 15 are typical of a Master that accesses the byte locations in ascending order. DTB protocols do not require these sequences.

As shown in Table 15, 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 being 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 Figure 10.

Table 15 – 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 Figure 9)
A	Quad-byte transfer	D[31..0]	Grp 1, Byte(0-3)
B	Single-byte transfer Double-byte transfer Single-byte transfer or Triple-byte transfer Single-byte transfer	D[7..0] D[15..0] D[15..8] D[23..0] D[15..8]	Grp 1, Byte(1) Grp 1, Byte(2-3) Grp 2, Byte(0) Grp 1, Byte(1-3) Grp 2, Byte(0)
C	Double-byte transfer Double-byte transfer	D[15..0] D[15..0]	Grp 1, Byte(2-3) Grp 2, Byte(0-1)
D	Single-byte transfer Double-byte transfer Single-byte transfer or Single-byte transfer Triple-byte transfer	D[7..0] D[15..0] D[15..8] D[7..0] D[31..8]	Grp 1, Byte(3) Grp 2, Byte(0-1) Grp 2, Byte(2) Grp 1, Byte(3) Grp 2, Byte(0-2)

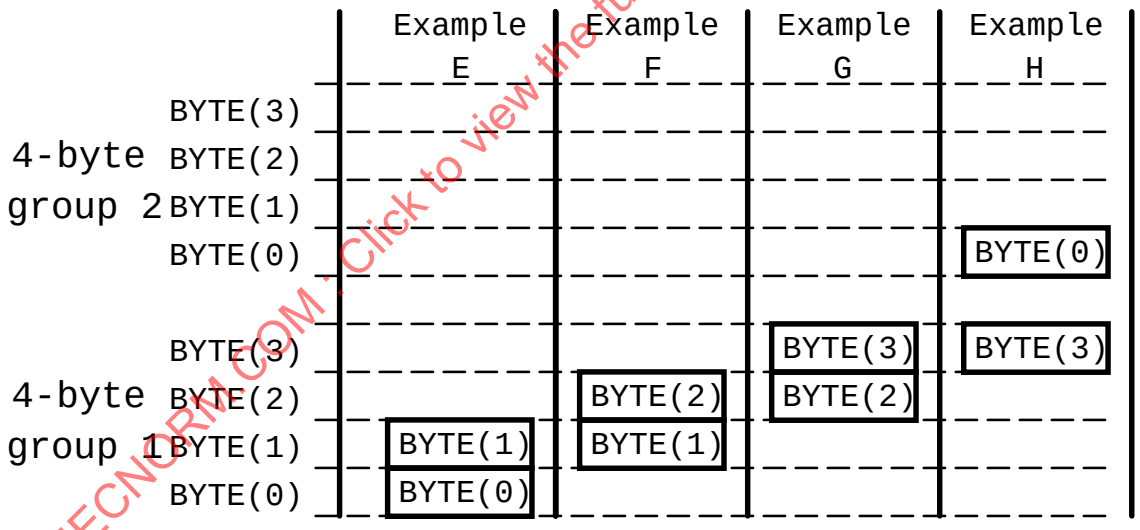


Figure 10 – Four ways in which 16 bits of data might be stored in memory

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

OBSERVATION 2.23

The sequences listed in Table 16 are typical of a Master that accesses the byte locations in ascending order. The VMEbus protocol do not require this.

As shown in Table 15, 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.

OBSERVATION 2.24

Since unaligned transfers make use of all 32 data lines, only D32 Masters and D32 Slaves can do unaligned transfers.

Table 17 lists how the Unaligned Transfer mnemonic (UAT) is used to describe Masters, Slaves, and Location Monitors.

Table 16 – 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 Figure 12)
E	Double-byte transfer	D[15..0]	Grp 1, Byte(0-1)
F	Single-byte transfer Single-byte transfer or Double-byte transfer	D[7..0] D[15..8] D[23..8]	Grp 1, Byte(1) Grp 1, Byte(2) Grp 1, Byte(1-2)
G	Double-byte transfer	D[15..8]	Grp 1, Byte(2-3)
H	Single-byte transfer Single-byte transfer	D[7..0] D[15..8]	Grp 1, Byte(3) Grp 2, Byte(0)

RECOMMENDATION 2.6 (*used to be RULE 2.67*)

D32 Masters, Slaves and Location Monitors should support UAT capability when appropriate.

RULE 2.6

A Slave **MUST NOT** respond with a falling edge on DTACK* during an unaligned transfer cycle, if it does not have UAT capability.

Table 17 – Mnemonic specifying 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
	D32 Slave	can accept 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
	D32 Location Monitor	can monitor 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 (ADO) on the DTB is not used to transfer data, but can be used to transfer special coded information. 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 ADO cycle is also the only type of DTB cycle that does not require a response to complete the cycle.

A second type of address-only cycle is ADDRESS-ONLY-WITH-HANDSHAKE (ADOH). 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. The data strobes are also driven low. After a Slave recognizes the address, it asserts DTACK*, BERR* or RETRY* low in response to the specific request. The WRITE* line may or may not be asserted during an ADOH cycle. The ADOH cycle is typically used for the lock commands.

Table 18 shows how the mnemonics ADO and ADOH are used to describe Masters and Slaves.

OBSERVATION 2.25

Address-only cycles (ADO) 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.

NOTE This capability was included for historical reasons related to the use of certain CPUs in use at the time. This practice is not recommended.

Table 18 – Mnemonics specifying address-only capability

The following mnemonic	when applied to a	means that it
ADO	Master	can generate Address-Only cycles
ADOH	Master	can generate Address-Only -With-Handshake cycles
	Slaves	can accept Address-Only-With-Handshake cycles

RULE 2.68

All Slaves **MUST** be designed to accommodate ADO cycles without loss of data or erroneous operation.

PERMISSION 2.15

All Masters MAY generate address-only cycles (ADO) with no handshake.

OBSERVATION 2.94

In all ADO cycles, neither of the two data strobes (DS0* and DS1*) are asserted low.

2.3.11 Lock capability

The lock commands are Address-Only-Cycles-With-Handshake (ADOH) cycles that are used to lock the other ports of multiported resources, where one port is on VMEbus. Each Slave's resource that is addressed with a lock command is to lock out all other accesses to that resource. The resource may be dual port memory, registers, special flags or control bits. Note that most Slaves have multiple resources. Only the resource that is addressed is to be locked and the other resources should remain unlocked. The extent of the locked resource is implementation dependent.

Each of the lock commands has an address phase with the same protocol as MBLT's address phase. The address (A16, A24, A32, A40 or A64) is presented to the bus and fully handshaked with the targeted Slave.

All locked resources are to remain locked until the end of the Master's bus mastership. End of bus mastership is signaled by the release of both BBSY* and AS* going high. This allows a Master to lock multiple ports in one lock sequence.

Use of a Lock Command signifies the start of a locked sequence. Each Slave that was addressed with a lock command records the start of the locked sequence. All subsequent transfers within the Master's mastership are guaranteed to be indivisible with assurance that no other operation can be performed on that resource. The locked sequence ends at the end of the current Master's bus mastership.

The effect of BERR* or RETRY* assertion is to end bus mastership and therefore unlock any locked resources. Error recovery and retry operations are not specified and are left up to higher level protocols.

A Master with its bus requester configured for ROR (release-on-request) requires a special operation. After performing a bus operation the Master effectively parks on the bus by not releasing the bus since no other Master wants to use the bus. If the Master ran any locked cycles it should release the bus even though no other Master wants the bus. This will ensure that all locks are released after any locked sequence is performed. The Master can optionally request to use the bus again.

RULE 2.81

Slaves that accept a lock command **MUST** lock out all other accesses to that resource until the bus is released.

RULE 2.82

Slaves **MUST** unlock any locks when bus mastership is released, which is signaled by both AS* and BBSY* high.

RECOMMENDATION 2.7

Slaves should drive RETRY* low in response to a lock command if any of its other port(s) have locked out bus accesses to the selected resource.

RULE 2.83

A Master **MUST** release BBSY* high after the last rising edge of AS* when it finishes its locked sequence.

OBSERVATION 2.95

Rule 2.83 precludes the early release of BBSY* during the last data transfer of a locked transfer.

RULE 2.84

All lock commands **MUST** be terminated with DTACK*, BERR*, or a RETRY* being driven low.

OBSERVATION 2.96

A locked Slave can only be unlocked when the bus is released. Parking on the bus MAY maintain the lock beyond the point at which it was intended to be released.

RECOMMENDATION 2.8

Masters that can generate lock commands should be labeled as A16:LCK, A24:LCK, A32:LCK, A40:LCK or A64:LCK.

RECOMMENDATION 2.9

Slaves that can accept and support lock commands should be labeled as A16:LCK, A24:LCK, A32:LCK, A40:LCK or A64:LCK.

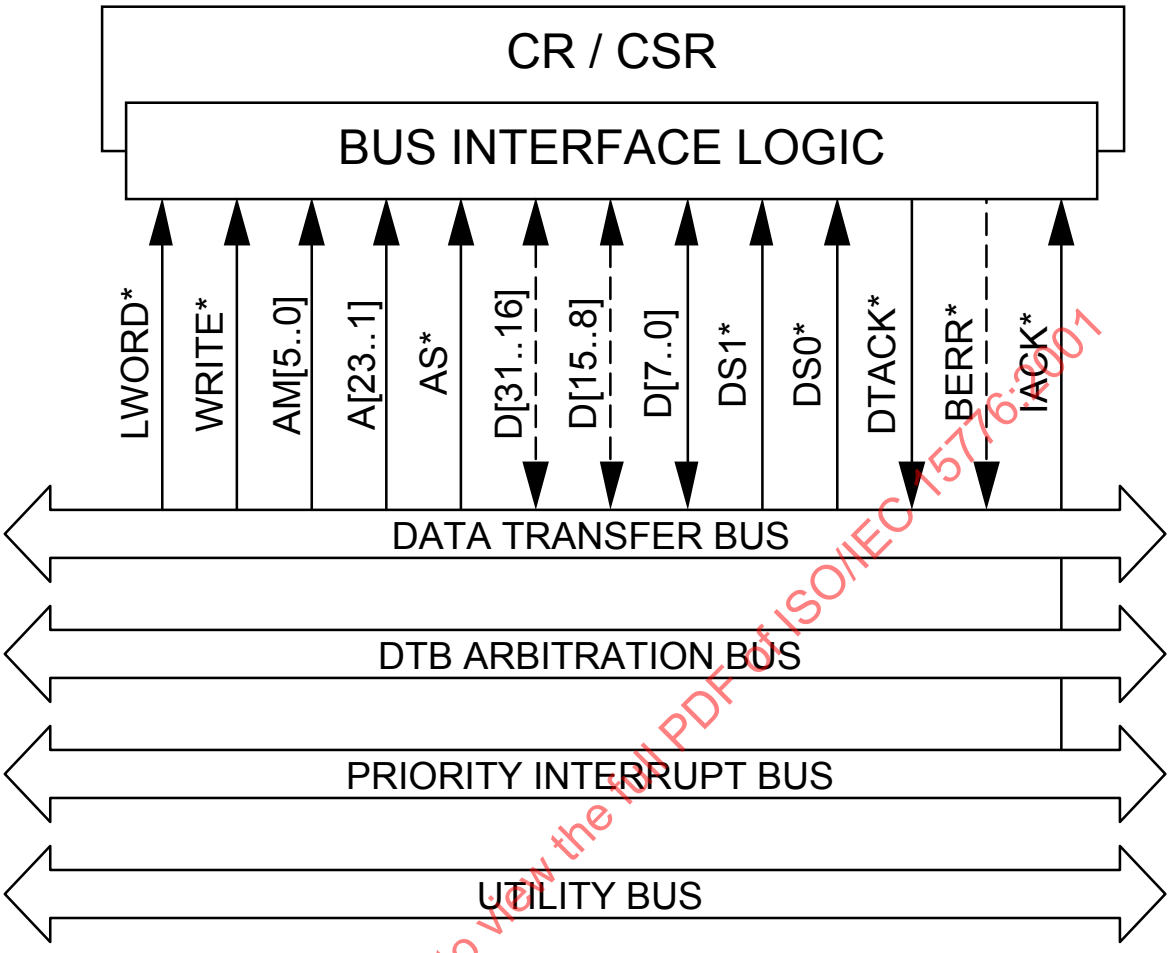
2.3.12 Configuration ROM/Control and status register capability

The Configuration ROM/Control and Status Registers (CR/CSR) provide a mechanism for manufacturer identification, board identification, automatic board initialization, board test and board configuration.

The CR/CSR space is accessed using a specific address modifier code (refer to table 4). Accesses to CR/CSR space use the A24 address mode to provide compatibility between 6U and 3U boards. The CR/CSR space is divided into 32 independent 512 KBytes regions. Each board in a system may occupy one of these CR/CSR regions. The method used for establishing unique addresses for each board is not defined. The method can be jumpers, Autoslot ID, etc. The 5 most significant bits of the A24 address range must be unique, A(23..19). Address lines, A(18..1), are used to address within each CR/CSR region. Data width modes D08(O), D08(EO), D16 and D32 are used to access a board's CR/CSRs. 3U boards are limited to D08(O), D08(EO) and D16 data width modes.

The lower portion (lowest address) of each CR/CSR region is to be used for configuration ROM and the upper portion (highest address) is to be used for a board's control and status registers (CSRs).

The CR/CSR block diagram is shown in Figure 11. The dashed lines show signals whose use varies among the various implementations of CR/CSRs. Table 19 specifies the requirements to monitor these lines.



NOTE Accessing CR/CSR is via a slave interface. All appropriate Slave rules, recommendations, observations, and timing parameters are to be used for CR/CSR accesses.

Figure 11 – Block diagram – Configuration ROM/Control & Status registers

RULE 2.85

Each board with CR/CSR capability **MUST NOT** occupy more than one 512 KB CR/CSR region. The block **MUST** be divided into two portions, with the first portion being a configuration ROM area and the second portion a CSR area. The Configuration ROM **MUST** start at the bottom (lowest) address. The CSR **MUST** start at the highest address and work its way towards the configuration ROM. Table 20 defines the CSR Base Register.

RULE 2.86

Bytes 0x00 through 0x7F of the Configuration ROM **MUST** be programmed as defined in Table 21.

Table 19 – Configuration ROM/Control & Status registers – RULEs and PERMISSIONs or monitoring dashed lines

Type of CR/CSR	RULEs and PERMISSIONs
D08(O)	**MAY** monitor D[31..8]
D08(EO)	**MUST** monitor D[15..8] **MAY** monitor D[31..16]
D16	**MUST** monitor D[15..8] **MAY** monitor D[31..16]
D32	**MUST** monitor D[31..8]
All	**MAY** drive BERR*
NOTE The mnemonics D08(E), D08(EO), D16 and D32 are defined in Table 12.	

RULE 2.87

The minimum Configuration ROM access data width **MUST** be D08(O), with access to every fourth byte, starting at byte 0x03.

PERMISSION 2.16

Boards may be designed without a CR/CSR capability.

Table 20 – Control and status register base definition

Address	Offset	Definition
0x7FFFF	CR/CSR Base Address Register (BAR). The BAR is used to select one of the 31 available CR/CSR regions (region 0x00 is reserved for use in Auto Slot ID). Thus the BAR values are in the range 1 to 31 (decimal). The BAR bits 7 to 3 define the CR/CSR region and correspond to A[23..19] in CR/CSR space.	
0x7FFFB	Bit Set Register	
	On Writes	On Reads
Bit 7	1: put board in reset mode	1: board is in reset mode
	0: no effect	0: board is not in reset mode
Bit 6	1: enable SYSFAIL driver	1: SYSFAIL driver is enabled
	0: no effect	0: SYSFAIL driver not enabled
Bit 5	1: no effect	1: board has failed
	0: no effect	0: board has not failed
0x7FFF7	Bit Clear Register	
	On Writes	On Reads
Bit 7	1: remove board from reset mode	1: board is in reset mode
	0: no effect	0: board is not in reset mode
Bit 6	1: disable SYSFAIL driver	1: SYSFAIL driver is enabled
	0: no effect	0: SYSFAIL driver not enabled
Bit 5	1: no effect	1: board has failed
	0: no effect	0: board has not failed
* Setting Bit 7 (RESET) must not affect normal VMEbus operation (i.e. daisy chains and all the bus protocols are unaffected. System Controller functions are also unaffected by the RESET bit). Further definition of reset mode is implementation dependent. Except as previously defined, locations 0x7FFF0 through 0x7FFFF are reserved.		

Table 21 – Configuration ROM definition

Address	Offset	Definition
0x03	Checksum. An eight bit 2s complement binary checksum. The *sum of bytes* is from offset 0x07 for the number of bytes specified in the length field (inclusive) allowing for the CSR data access width. Area to address 0x7F is assumed to be every fourth byte regardless of the ROM data access width specified in 0x13.	
0x07, 0x0B, 0x0F	Length of ROM that is to be checksummed. The length is a three byte 0x0F binary value with the MSB in byte 0x07. Note that bits 7 through 3 of the MSB should be all zeros. The range of length is 0x1F to 0x7FFF0.	
0x13	Configuration ROM data access width:	
	0x00	Not to be used
	0x01-0x80	Reserved for future use
	0x81	Only use D08(EO), every fourth byte
	0x82	Only use D08(EO), every other byte
	0x83	Use D16 or D08(EO), every byte used
	0x84	Use D32, D16 or D08(EO), every byte used
	0x85-0xFE	Reserved for future use
	0xFF	Not to be used
0x17	CSR data access width:	
	0x00	Not to be used
	0x01-0x80	Reserved for future use
	0x81	Only use D08(EO), every fourth byte
	0x82	Only use D08(EO), every other byte
	0x83	Use D16 or D08(EO), every byte used
	0x84	Use D32, D16 or D08(EO), every byte used
	0x85-0xFE	Reserved for future use
	0xFF	Not to be used
0x1B	CSR /CSR Space Specification ID	
	0x00	Not to be used
	0x01	VME64 -1994 version
	0x02-0xFE	Reserved for future use
	0xFF	Not to be used
0x1F	0x43	'C'. Used to identify a valid CR
0x23	0x52	'R'. Used to identify a valid CR
0x27, 0x2B, 0x2F	Manufac-turer's ID	The 24 bit ID assigned by IEEE. 0x27 is the MSB, 0x2F is LSB. See annex C.
0x33, 0x37, 0x3B & 0x3F	Board ID (4 bytes) (Binary number, supplied & controlled by manufacturer)	
0x53, 0x57 & 0x5B	Pointer to a null terminated ASCII type printable string or a null value (0x000000) if no string is available. The pointer is relative to a base address of 0x000000 and is assumed to be in the CR address region.	
0x5F to 0x7B	Reserved for future use	
0x7F	Program ID Code	
	0x00	Not used
	0x01	No program, ID ROM only
	0x02-0x4F	Manufacturer defined
	0x50-7F	User defined
	0x80-EF	Reserved for future use
	0xF0-FE	Reserved by Boot Firmware (P1275)
	0xFF	Not to be used
0x80	Start of user defined area, including special programs.	

OBSERVATION 2.97

The method used to configure a module and the system via the CR/CSR is beyond the scope of this standard.

RULE 2.88

Accessing CR/CSRs is the same as accessing a slave and **MUST** follow the same rules as specified for slaves.

OBSERVATION 2.98

All Configuration ROM byte locations between 0x00 and 0x7F not defined in Table 21 are reserved.

RECOMMENDATION 2.10

All reserved locations in the Configuration ROM should read as either 0x00 or 0xFF.

Attention is drawn to the fact that “Boot Firmware” accesses only one byte at a time. Therefore, the Configuration ROM interface implementation may be simplified by using only a byte wide ROM/PROM.

The definition and usage of the control and status registers (CSR) is user defined. Interface access to CSRs can be any data width: D08(O), D08(EO), D16 and/or D32. CSRs are to be placed at the top end of the board's CR/CSR region. This allows for up to 512 KB of Configuration ROM, minus the area used for CSRs.

2.3.13 Retry capability

RETRY* can be driven low by a responding Slave to indicate to the Master that the requested data transfer cannot be executed at this time. The Master that supports Retry should retry the data transfer at a later time. In the interim, the bus can be used for other transactions.

If a Slave supports Retry capability, it can assert RETRY* low to signal the Master that either the requested resource is busy or that a deadlock condition has occurred. If the Master supports retry capability, it terminates the bus cycle when it detects RETRY* low without waiting for either DTACK* or BERR*.

In the case of deadlock condition, if the bus Master does not support Retry and does not terminate the cycle, the Slave drives BERR* to end the bus cycle.

In the case of a busy condition, if the Master does not support Retry and does not terminate the cycle the slave waits and asserts DTACK* low once the busy resource becomes available.

Figure 35 shows the timing for the case when the Master terminates the bus cycle when it detects RETRY* low. Figure 36 shows the timing for the case when the Master ignores RETRY* low. Additional RULEs and OBSERVATIONs regarding the use of RETRY* signal are covered in Tables 29, 30, and 31.

PERMISSION 2.17

VMEbus Masters and Slaves MAY be designed without support for RETRY* signal.

OBSERVATION 2.99

The bus cycle is always terminated with either DTACK* low or BERR* low if either the Master or the selected Slave does not support retry capability.

RULE 2.89

The RETRY* driver **MUST** be a rescinding driver.

RULE 2.90

If the Master responds to a RETRY* low, then it **MUST** release the bus and allow another Master to use the bus if there is another request pending, regardless of the request level.

RULE 2.91

If a Slave wishes to RETRY* a block transfer cycle it **MUST** do so during the address phase of the cycle; not the data phase.

PERMISSION 2.18

After releasing control of the DTB as a result of detecting RETRY* low, the Master MAY signal its requester to request the bus again.

RECOMMENDATION 2.11

After releasing the control of the DTB as a result of detecting RETRY* low, the Master should wait for a user-defined period of time before requesting the bus again, if there are other requests pending.

2.3.14 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 2.26a

Although the address and data timing are largely independent, there are two exceptions. First, the Master waits until it has driven AS* low before driving either of the data strobes low. Second, the Slave acknowledges both the address strobe and the data strobes with either DTACK*, RETRY* or BERR*.

RULE 2.13a

IF a Slave responds to a data transfer cycle with DTACK* or BERR*,

THEN it **MUST** either drive DTACK* low or it **MUST** drive BERR* low, but not both.

OBSERVATION 2.27

Because of possible bus skew, due to different loading of the address strobe and the data strobes, the falling edge of the data strobes might be detected by the Slave slightly before the address strobe falling edge.

RULE 2.14a

Before driving the data bus, the Master **MUST** ensure that the previous responding Slave has stopped driving the data bus. It does so by verifying that DTACK*, and BERR* are high before it drives the data strobe(s) to low on any cycle, and before it drives any of the data lines during a write cycle.

RULE 2.15a

At the end of a read cycle and at the end of the last read of a block read cycle, the responding Slave **MUST** release the lines that carry data before allowing DTACK* or RETRY* to go high or driving DTACK* or RETRY* to high.

RULE 2.16

When the Master reads data from the Slave, the Slave **MUST** maintain valid data on the data bus until the Master returns the first data strobe to high.

SUGGESTION 2.3a

For optimum performance, Masters have to be designed in such a way that they drive the data strobes high as soon as possible after DTACK*, BERR*, or RETRY* goes low. Likewise, Slaves have to be designed in such a way that they release the data bus, BERR*, RETRY*, and DTACK* high as soon as possible after detecting that the data strobes high. This allows the maximum data transfer rate on the bus. (Also refer to OBSERVATION 2.102.)

RECOMMENDATION 2.12

For higher performance, it is desirable to drive DTACK* high for 30 ns or less before releasing DTACK*.

RULE 2.92

Masters that drive DS0* and/or DS1* and AS* high at the end of a cycle, **MUST** do so for no longer than 30 ns, if the Master also releases bus mastership.

OBSERVATION 2.29

Addressing information on the bus might change soon after a module drives DTACK* or BERR* low, and before the Master drives the data strobes high.

A third type of module, the Location Monitor, monitors the data transfer and generates on-board signal(s) whenever a location that it monitors is accessed.

If the cycle takes too long, a fourth module, called a Bus Timer intervenes by driving BERR* low, completing the data transfer handshake, and allowing the bus to resume operation.

RULE 2.17a

There is a strict interlock between the rising and falling edges of the data strobes and DTACK*/BERR*/RETRY*. Once a Master has driven its data strobe(s) low it **MUST NOT** drive its data strobe(s) high and finish a transfer without first receiving a data transfer acknowledge, a bus error response, or a retry request.

OBSERVATION 2.30a

A board containing a processor, which needs to direct data transfers between itself and other VMEbus boards, would contain a Master module. If the same board also contained memory accessible from the VMEbus, it would also contain a Slave module. A floating point processor or 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 VMEbus 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 doing 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 (see 2.5 for further explanations). The following discussion presumes that the Master has already been granted and has assumed control of the DTB.

2.4.1 Typical data-transfer cycles

Figure 12 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 A1 low. Since it is not 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. If 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 D[7..0], 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 waits the prescribed amount of time and then latches the data on D[7..0], releases the address lines and drives DS0* and AS* high. The Slave responds by releasing D[7..0] and releasing DTACK* high.

OBSERVATION 2.31

The Master in Figure 12 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 cycle-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 Figures 13 and 14.

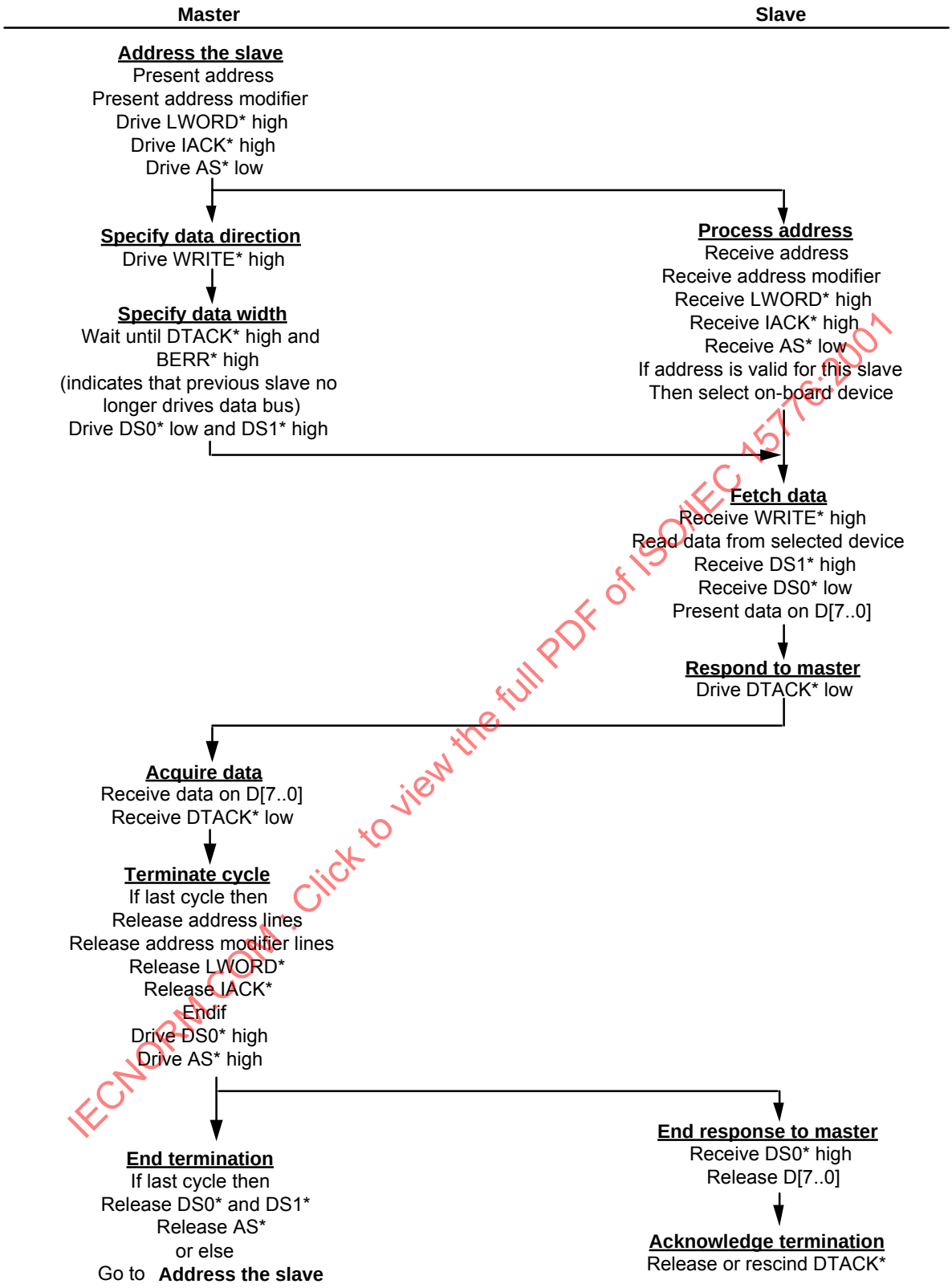


Figure 12 – Example of non-multiplexed address single-byte read cycle

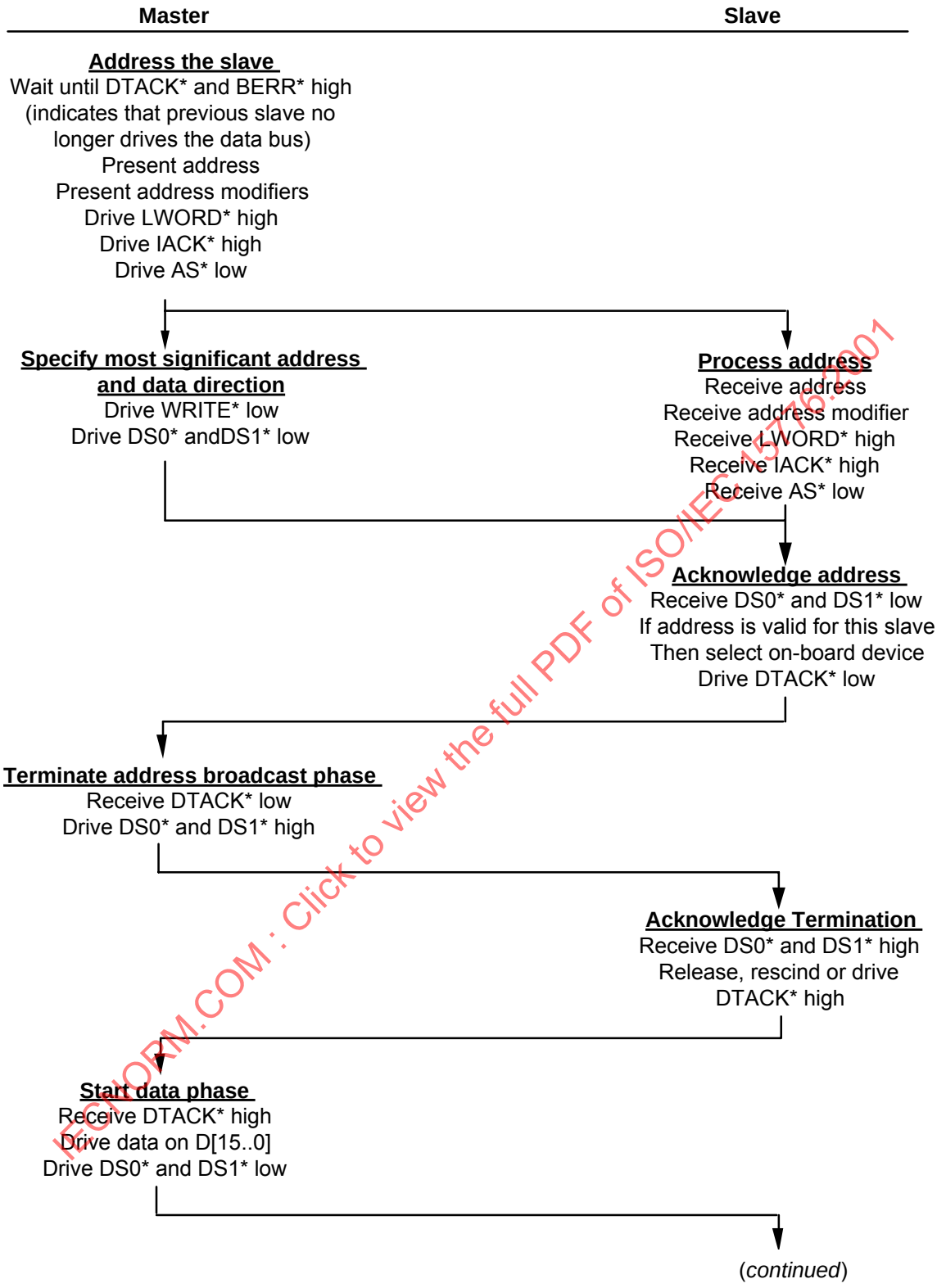


Figure 13 – Example of multiplexed address double-byte write cycle

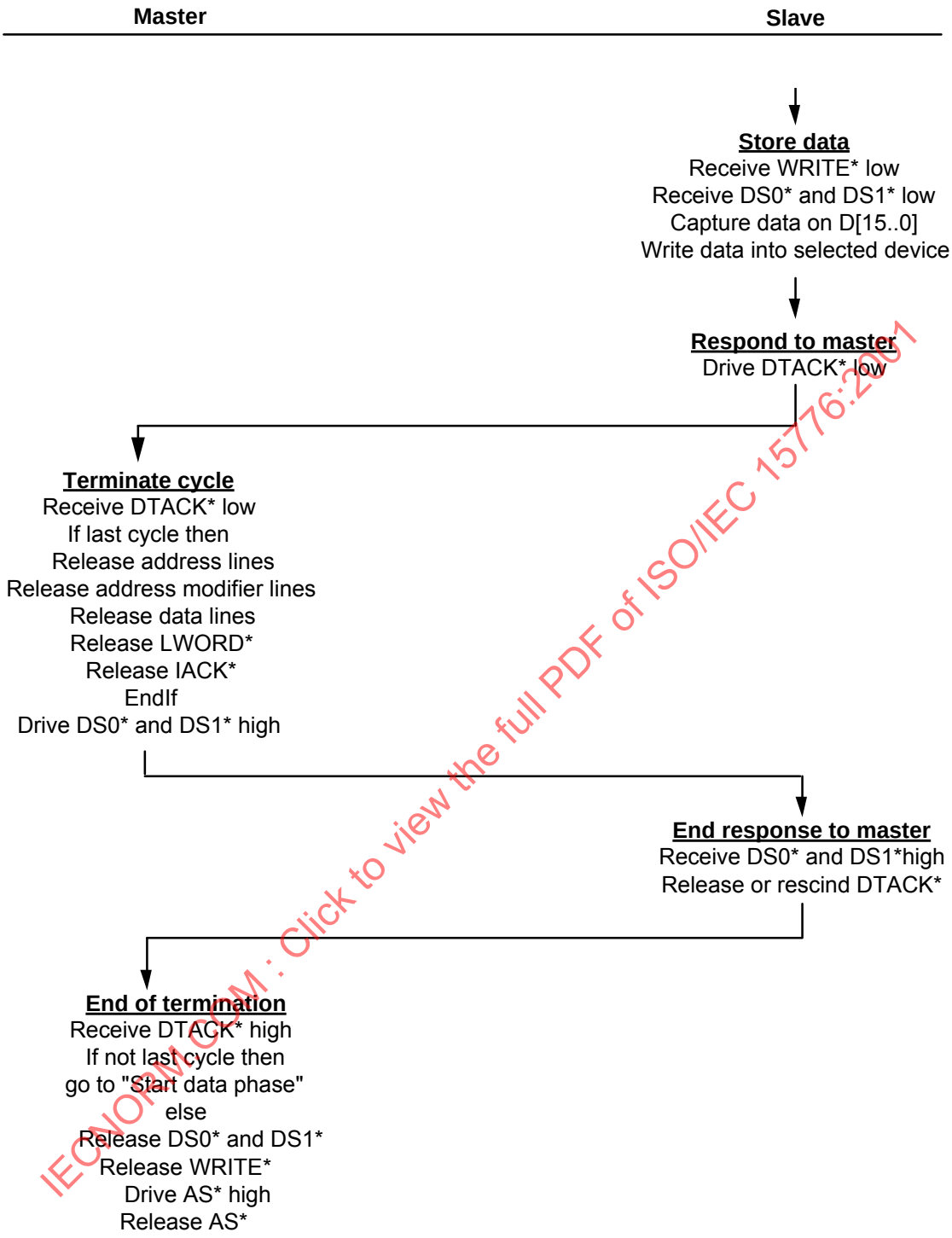


Figure 13 – Example of multiplexed address double-byte write cycle *(continued)*

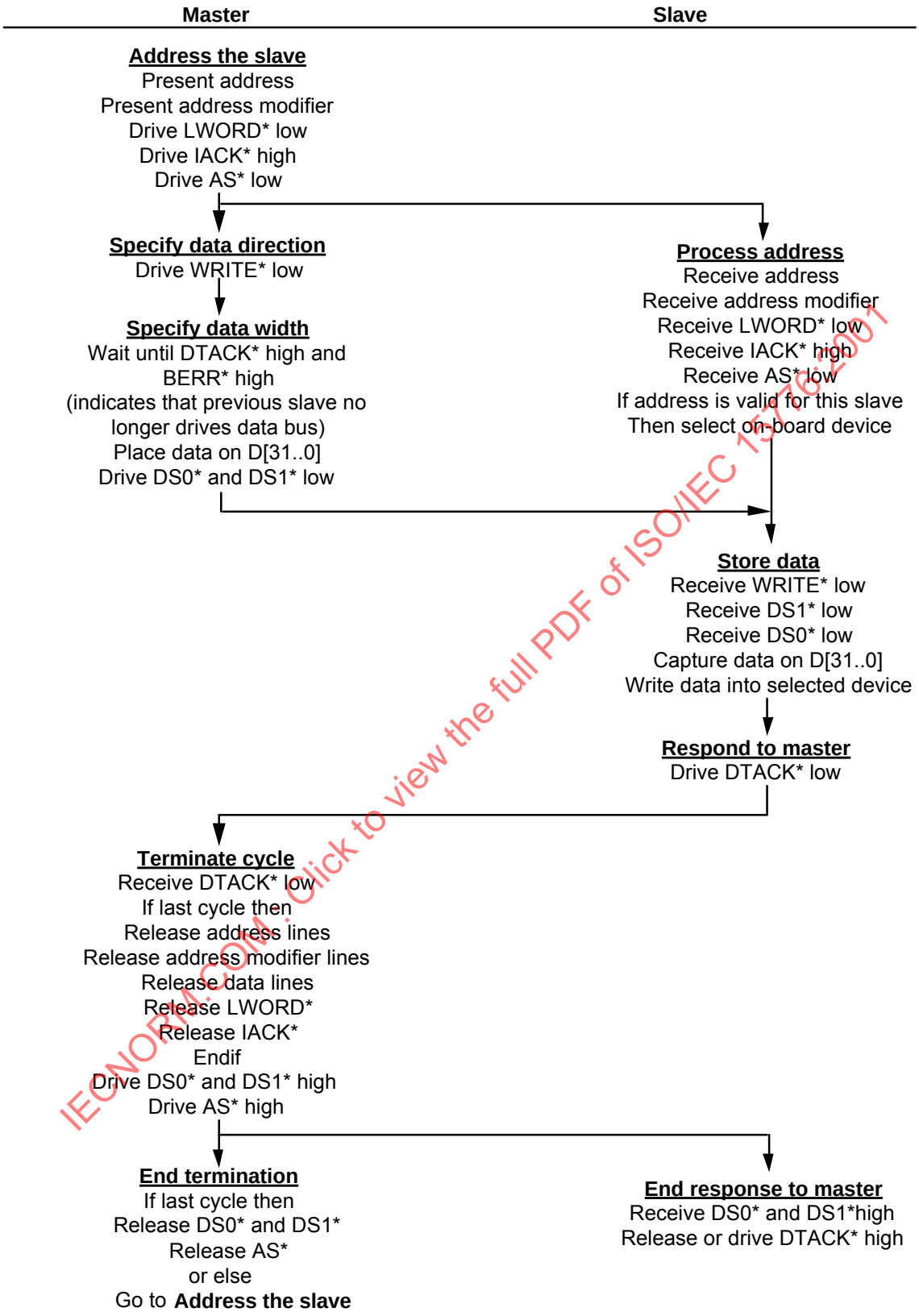


Figure 14 – Example of non-multiplexed address quad-byte write cycle

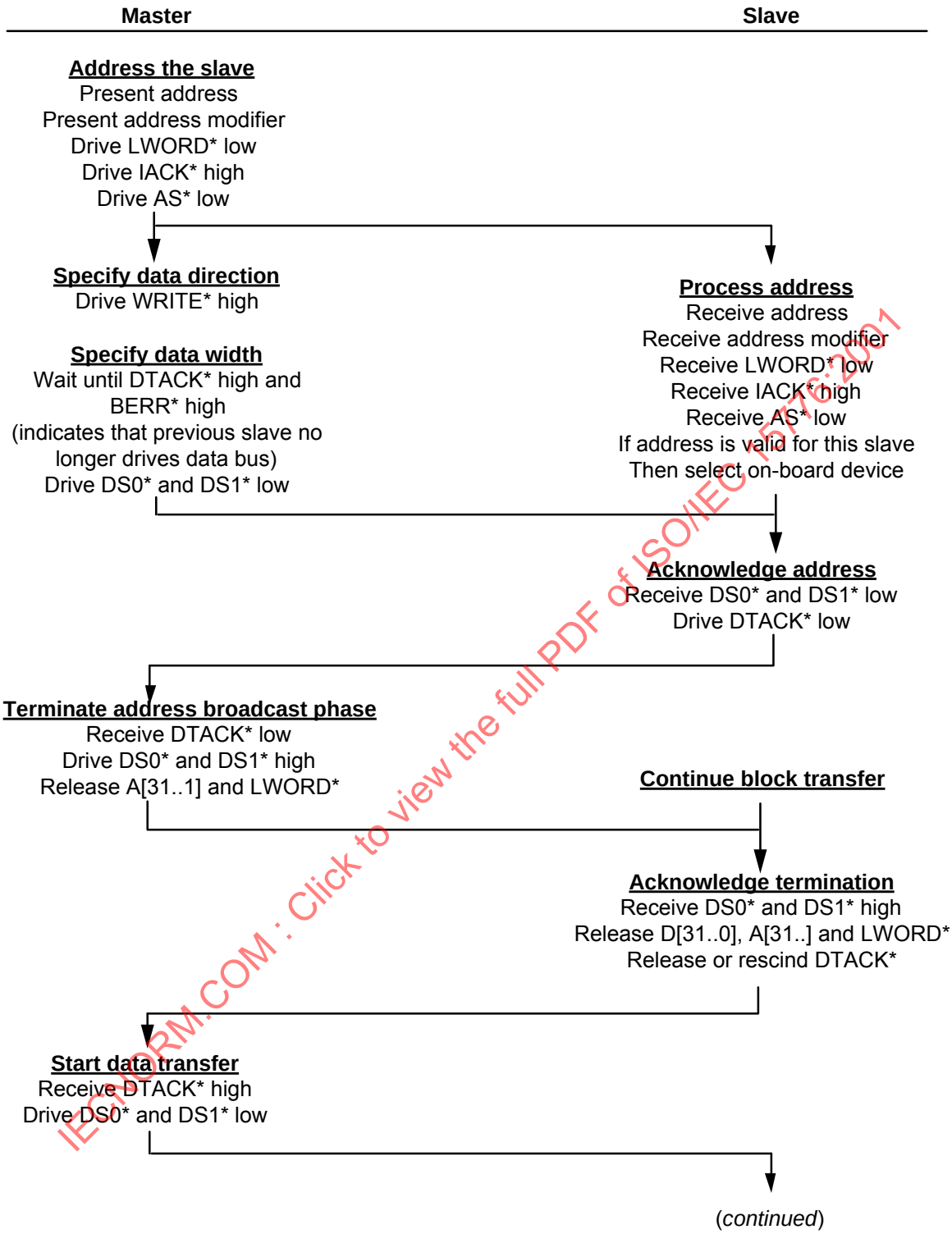


Figure 15 – Example of eight-byte block read cycle

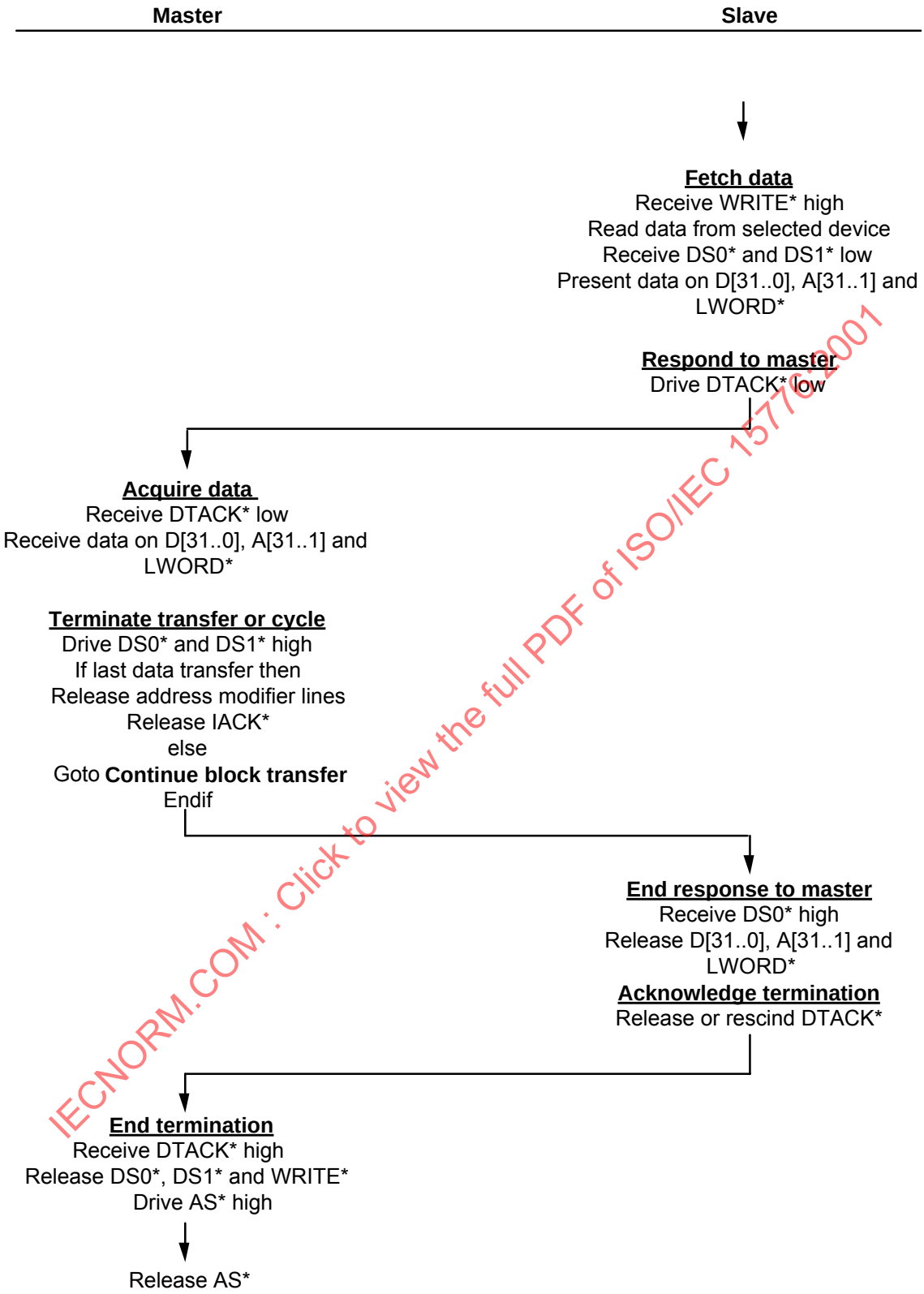


Figure 15 – Example of eight-byte block read cycle *(continued)*

2.4.2 Address pipelining

The bus strobes the address and data with separate strobe signals. This allows a Master to 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 2.8a

During all cycles except A40, A64, MD32 and MBLT cycles, 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 can 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 2.93

Masters **MUST NOT** use address pipelining during A40, A64 or MBLT and MD32 transfers.

OBSERVATION 2.100

During MBLT and MD32 read cycles, Slaves use address lines to transfer data. Therefore, the address for the next cycle cannot be broadcast until after the responding Slave releases DTACK* and BERR*, signifying that it no longer drives the address lines.

OBSERVATION 2.101

During A64 and A40 address phases, Masters use both the address and data lines to broadcast the address. Therefore, the address for the next cycle cannot be broadcast until after the responding Slave releases DTACK* and BERR*, signifying that it no longer is driving the data lines.

RULE 2.18

Since address pipelining can occur, Slaves **MUST NOT** be designed on the assumption that they will never encounter pipelined cycles.

OBSERVATION 2.32

The responding Slave may 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.

SUGGESTION 2.4

Design all Slaves to monitor AS*, and to capture (latch) the addressing information from A[31..1] and LWORD* on the falling edge of AS*.

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 if that address is needed by on-board logic to maintain the data on the bus.

PERMISSION 2.9

Masters MAY be designed without the ability for address pipelining. (E.g. they MAY wait until the responding Slave releases DTACK* or BERR* before driving AS* low for the next cycle.)

OBSERVATION 2.34

A Master might drive AS* low for a new cycle before it drives data strobes high from the previous cycle. Because of this, there might be a period when the address strobe for the new cycle, as well as at least one data strobe from the previous cycle coincide during the cycle overlap.

SUGGESTION 2.5a

Design Slaves to respond to data transfers when one or both of DS0* and DS1* are low and when DTACK* and BERR* are both high, instead of a simultaneous low level on AS* and one or both of DS0* and DS1*.

2.5 Data-transfer-bus acquisition

RULE 2.19

Before transferring any data on the DTB, a Master **MUST** get permission to use it.

Several Masters might want to use the DTB at the same time. The process that determines which Master can use the DTB is called arbitration and is discussed in clause 3. Because arbitration is closely tied to the operation of the DTB, it is briefly described here.

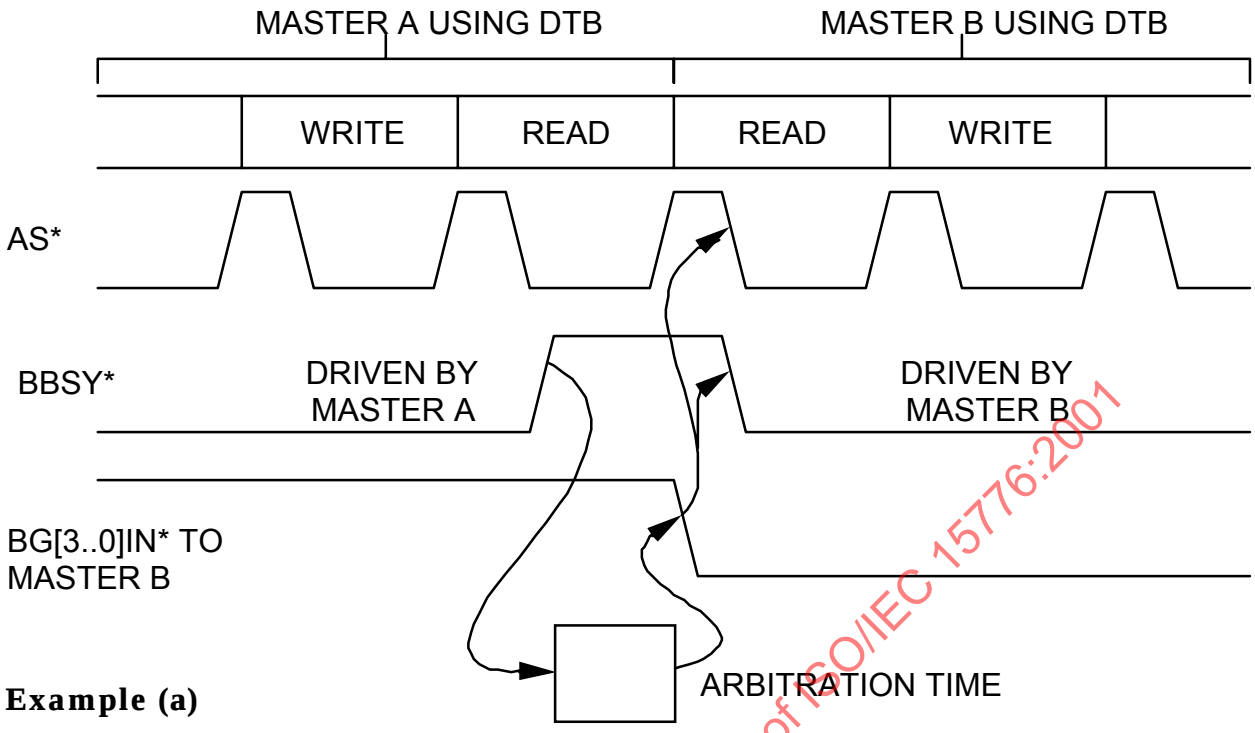
Figure 15 provides two examples that show possible sequences when a Master (called Master A) finishes using the DTB and allows arbitration to take place.

In Example 1, Master A, part way into its last transfer, indicates that it no longer needs the DTB. It does this by having its Requester release the bus busy (BBSY*) signal line. Since Master A gives this early notice that the DTB will soon be available, the arbitration takes place during the last data transfer. The arbitration is completed and Master B is granted permission to use the DTB before Master A has terminated its cycle, but it waits until Master A releases AS*. (This ensures that Master B will not start driving the DTB before Master A has finished with its last data transfer.)

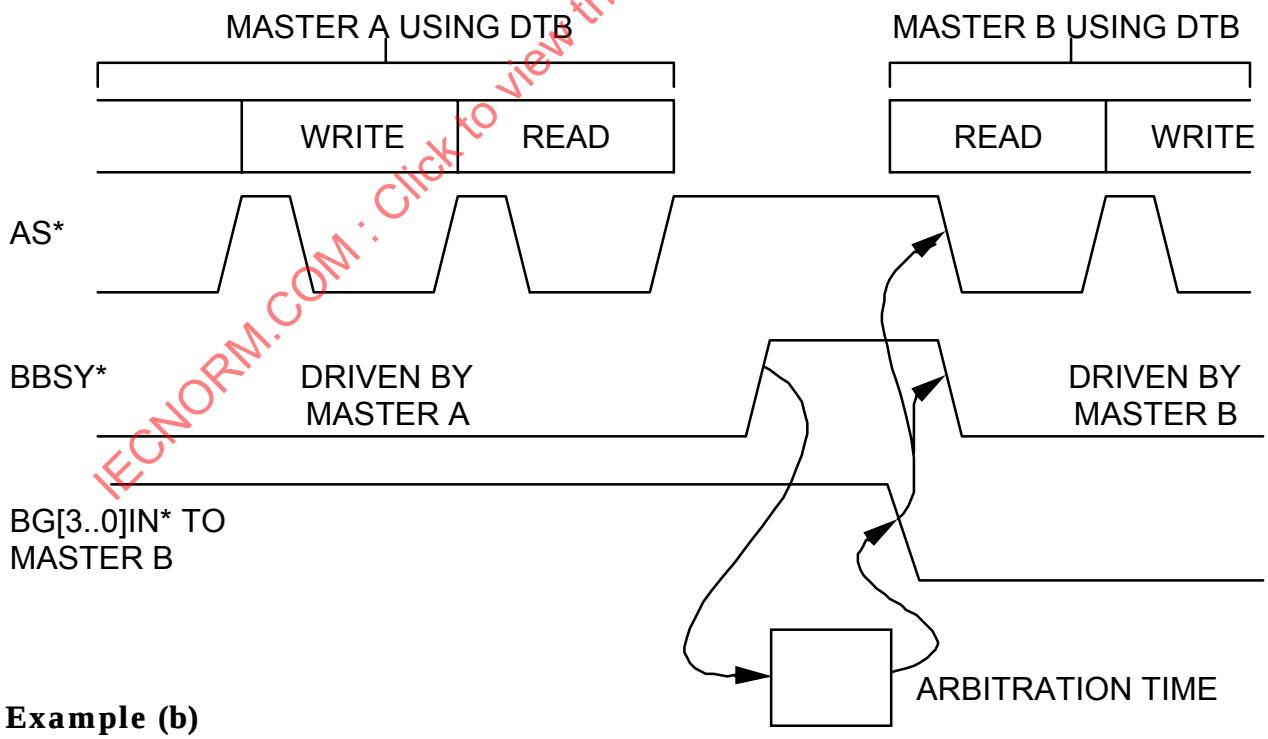
In Example 2, Master A waits until after its last transfer (i.e., after AS* is released) before releasing BBSY*. In this case the DTB is idle while the arbitration is done. Master B is then granted the bus and, since AS* is already high, it begins using the DTB immediately.

RULE 2.20

Once a Master's Requester releases BBSY* to high, the Master **MUST NOT** drive AS* from high to low (i.e., it **MUST NOT** begin a new cycle) until its Requester receives a new bus grant.



Example (a)



Example (b)

Figure 16 – Data transfer bus master exchange sequence

2.6 DTB timing rules and observations

The timing rules and observations that govern the behavior of Masters and Slaves are being described in this subclause. This information is given in the Figures and tables as specified below.

Table 22 lists a timing table and timing diagrams specifying Master, Slave, and Location Monitor operation.

Table 23 defines the various mnemonics that are used in Tables 24 through 26.

Tables 24 through 26 specify the use of the DTB signals.

Tables 27 through 32 specify the timing parameters of the DTB. (The reference numbers used in Tables 29 through 32 correspond to the timing parameter numbers in Tables 27 and 28.)

Figures 17 through 21 specify the timing RULEs and OBSERVATIONs during address broadcasting time.

Figures 22 through 29 and 37 specify the timing RULEs and OBSERVATIONs for Masters, Slaves, and Location Monitors during data transfer time.

Figures 30 through 32 specify the timing RULEs and OBSERVATIONs for Masters and Slaves between DTB cycles.

Figure 33 is the timing diagram for Master, Slave, and Bus Timer during a timed-out cycle.

Figure 34 shows the timing during the mastership transfer of the DTB.

Figures 35 and 36 specify the timing RULEs and OBSERVATIONs for Masters and Slaves when RETRY * is used.

Figure 38 shows the timing for Rescinding DTACK*.

In order to meet the timing RULEs, board designers need to take into account the worst and best case propagation delays of bus drivers and receivers used on their VMEbus boards. The propagation delay of the drivers depends on their output loads, but manufacturer's specifications do not always give enough information to calculate the propagation delays under various loads. To help the board designer, some suggestions are offered in clause 6.

The OBSERVATIONs specify the timing of incoming line signal transitions. These times can be relied upon as long as the backplane loading RULEs in clause 6 are not violated. The RULEs for the bus terminators in clause 6 guarantee that the timing parameters for signal lines that are released after they have been driven, are met.

Typically, for each timing RULE there is a corresponding OBSERVATION. However, the time that is guaranteed in the OBSERVATION might differ from the time specified by the RULE. For example, a careful inspection of the timing diagrams shows that the Master is required to provide 35 ns of address and data set-up time, but the Slave is only guaranteed 10 ns. This is because the address and data bus drivers are not always able to drive the backplane's signal lines completely through the threshold region from low to high until the transition propagates to the end of the backplane and is reflected back. The falling edge of the address and data strobes, however, typically cross the 0,8 V threshold without waiting for a reflection. The resulting set-up time at the Slave is the Master's set-up time less two bus propagation times.

A special notation has been used to describe the data strobe timing. The two data strobes (DS0* and DS1*) will not always make their transitions simultaneously. For purposes of these timing diagrams, DSA* represents the first data strobe to make its transition (whether that is DS0* or DS1*). The broken line shown while the data strobes are stable is to indicate that the first data strobe to make a falling transition might not be the first to make its rising transition; i.e., DSA* can represent DS0* on its falling edge and DS1* on its rising edge.

Tables 24, 25, and 26 show how the various signal lines of the DTB are used to broadcast addresses and to transfer data. These tables are referenced by the various timing diagrams that follow. To keep these tables compact, mnemonics are used to describe when and how the various lines are driven. These mnemonics are defined in Table 23.

CR/CSR is a functional module defined in this clause, but is not specifically called out in this subclause. Access to CR/CSRs is treated as a slave access. Therefore all the appropriate slave RULES apply.

Table 22 – Timing diagrams defining Master, Slave, and LOCATION MONITOR operation
(See Table 27 for timing values)

Mnemonic	Type of cycles	Address broadcast timing Figure(s)	Data transfer timing Figure(s)
ADO	Address-Only	17	N/A
ADOH	Address-Only-With-Handshake	21	38
A40	A40 address broadcast	21	all
A64	A64 address broadcast	21	all
D08(EO)	Single-even-byte transfer		
	Byte(0) Read	17, 18, 21	22, 38
	Byte(2) Read	17, 18, 21	22, 38
	Byte(0) Write	17, 18, 21	25, 38
	Byte(2) Write	17, 18, 21	25, 38
D08(EO) or DO8(O)	Single-odd-byte transfers		
	Byte(1) Read	17, 18, 21	22, 38
	Byte(3) Read	17, 18, 21	22, 38
	Byte(1) Write	17, 18, 21	25, 38
	Byte(3) Write	17, 18, 21	25, 38
D16	Double-byte transfers		
	Byte(0-1) Read	17, 18, 21	23, 38
	Byte(2-3) Read	17, 18, 21	23, 38
	Byte(0-1) Write	17, 18, 21	26, 38
	Byte(2-3) Write	17, 18, 21	26, 38
D32	Quad-byte transfers		
	Byte(0-3) Read	17, 18, 21	23, 38
	Byte(0-3) Write	17, 18, 21	26, 38
MD32	Multiplexed quad byte transfers		
	Byte(0-3) Read	21	24, 38
	Byte(0-3) Write	21	27, 38
D08(EO): BLT	Single-byte block transfers		
	Single-Byte Block Read	17, 19, 21	24, 38
	Single-Byte Block Write	17, 19, 21	25, 38

Table 22 *(continued)*

Mnemonic	Type of cycles	Address broadcast timing Figure(s)	Data transfer timing Figure(s)
D16: BLT	Double-byte block transfers		
	Double-Byte Block Read	17, 21, 23	23, 38
	Double-Byte Block Write	17, 21, 23	26, 38
D32: BLT	Quad-byte block transfers		
	Quad-Byte Block Read	17, 21, 23	23, 38
	Quad-Byte Block Write	17, 21, 23	30, 38
MD32:BLT	Multiplexed quad byte block transfers		
	Multiplexed Quad-Byte Block Read	21	24, 38
	Multiplexed Quad-Byte Block Write	21	27, 38
MBLT	Eight-byte block transfers		
	Eight-Byte Block Read	21	24, 38
	Eight-Byte Block Write	21	27, 38
D08(EO) RMW	Single-byte RMW transfers		
	Byte(0) Read-Modify-Write	17, 20, 21	28, 38
	Byte(1) Read-Modify-Write	17, 20, 21	28, 38
	Byte(2) Read-Modify-Write	17, 20, 21	28, 38
	Byte(3) Read-Modify-Write	17, 20, 21	28, 38
D16: RMW	Double-byte RMW transfers		
	Byte(0-1) Read-Modify-Write	17, 20, 21	25, 38
	Byte(2-3) Read-Modify-Write	17, 20, 21	25, 38
D32: RMW	Quad-Byte RMW transfers		
	Byte(0-3) Read-Modify-Write	17, 20, 21	25, 38
MD32: RMW	Multiplexed Quad-Byte RMW transfers		
	Byte(0-3) Read-Modify-Write	21	37, 38
D32: UAT	Unaligned transfers		
	Byte(0-2) Read	17, 18, 21	22, 38
	Byte(1-3) Read	17, 18, 21	22, 38
	Byte(1-2) Read	17, 18, 21	23, 38
	Byte(0-2) Write	17, 18, 21	25, 38
	Byte(1-3) Write	17, 18, 21	25, 38
	Byte(1-2) Write	17, 18, 21	26, 38
N/A	Retry request	N/A	35, 36

Table 23 – Definitions of mnemonics used in tables 24, 25 and 26

Mnemonic	Description	RULES and PERMISSIONS
DVBM	Driven Valid by Master	**RULE 2.21** The Master **MUST** drive DVBM lines to a valid level.
DLBM	Driven Low by Master	**RULE 2.22** The Master **MUST** drive DLBM lines to a low level.
DHBM	Driven High by Master	**RULE 2.23** The Master **MUST** drive DHBM lines to a high level.
dhbm?	Driven High by Master?	**PERMISSION 2.10** The Master MAY drive dhbm? lines high **RULE 2.24** The Master **MUST NOT** drive dhbm? lines low.
dxbm?	Driven by Master?	**PERMISSION 2.11** The Master MAY drive dxbm? lines, or it may leave these lines undriven. (When dxbm? lines are driven, they carry no valid information.)
DVBS	Driven Valid by Slave	**RULE 2.25** The responding Slave **MUST** drive DVBS lines to a valid level.
dxbs?	Driven by Slave?	**PERMISSION 2.12** The responding Slave MAY drive dxbs? lines, or it may leave these lines undriven. (When dxbs? lines are driven, they carry no valid information.)
DVBB	Driven Valid by Both Slave and Master	**RULE 2.26** During the read portion of a Read-Modify-Write Cycle, the responding Slave **MUST** drive DVBB lines with valid data. During the write portion of a Read-Modify-Write Cycle, the Master **MUST** drive DVBB lines with valid data.
dxbb?	Driven by Both Slave and Master?	**PERMISSION 2.13** During the read portion of a Read-Modify-Write Cycle, the responding Slave MAY drive dxbb? lines, or it MAY leave them undriven. During the write portion of a Read-Modify-Write Cycle, the Master MAY drive dxbb? lines, or it MAY leave them undriven. (When dxbb? lines are driven they carry no valid information.)

Table 24 – Use of the address and data lines to select a byte group

Addressing mode	A[15..2] (Note 1)	A[23..16]	A[31..24]	D[31..16] (Note 2)	D[15..0] (Note 3)	IACK*
A16	DVBM	dxbm?	dxbm?	dxbm?	dxbm?	dhbm?
A24	DVBM	DVBM	dxbm?	dxbm?	dxbm?	dhbm?
A32	DVBM	DVBM	DVBM	dxbm?	dxbm?	dhbm?
A40	DVBM	DVBM	dxbm?	dxbm?	DVBM	dhbm?
A64	DVBM	DVBM	DVBM	DVBM	DVBM	dhbm?

NOTE 1 A1 is used in conjunction with LWORD*, DS0* and DS1* to select which of the bytes within the byte group is accessed (see Table 25).

NOTE 2 During the A16, A24 and A32 addressing modes, D[31..0] are not used to select byte-locations. They are multiplexed to carry the upper address bits only when using the A40 and A64 addressing modes.

Table 25 – Use of DS1*, DS0*, A1, A2, and LWORD* during the address phase of the various cycles

Mnemonic	Type of cycles	DS1*	DS0*	A1	A2	LWORD*
ADO	Address-Only	dhbm?	dhbm?	dxbm?	DVBM	dxbm?
ADOH	Address-Only-With-Handshake	DLBM	DLBM	DVBM	DVBM	DVBM
D08(EO)	Single-even-byte transfers					
	Byte(0) Read or Write	DLBM	dhbm?	DLBM	DVBM	dhbm?
	Byte(2) Read or Write	DLBM	dhbm?	DHBM	DVBM	dhbm?
D08(EO) or D08(O)	Single-odd-byte transfers					
	Byte(1) Read or Write	dhbm?	DLBM	DLBM	DVBM	dhbm?
	Byte(3) Read or Write	dhbm?	DLBM	DHBM	DVBM	dhbm?
D16	Double-byte transfers					
	Byte(0-1) Read or Write	DLBM	DLBM	DLBM	DVBM	dhbm?
	Byte(2-3) Read or Write	DLBM	DLBM	DHBM	DVBM	dhbm?
D32,MD32	Quad-byte transfers					
	Byte(0-3) Read or Write	DLBM	DLBM	DLBM	DVBM	DLBM
D08(EO) BLT	Single-byte block transfers					
	Single-Byte Block Read or Write	<	Note 1	>	DVBM	dhbm?
D16:BLT	Double-byte block transfers					
	Double-Byte Block Read or Write	DLBM	DLBM	Note 2	DVBM	dhbm?
D32:BLT, MD32:BLT	Quad-byte block transfers					
	Quad-Byte Block Read or Write	DLBM	DLBM	DLBM	DVBM	DLBM
MBLT	Eight-byte block transfers					
	Eight-Byte Block Read or Write	DLBM	DLBM	DLBM	DLBM	DLBM
D08(EO): RMW	Single-byte Read-Modify-Write transfers					
	Byte(0) RMW	DLBM	dhbm?	DLBM	DVBM	dhbm?
	Byte(1) RMW	dhbm?	DLBM	DLBM	DVBM	dhbm?
	Byte(2) RMW	DLBM	dhbm?	DHBM	DVBM	dhbm?
	Byte(3) RMW	dhbm?	DLBM	DHBM	DVBM	dhbm?
D16:RMW	Double-byteRead-Modify-Write transfers					
	Byte(0-1) RMW	DLBM	DLBM	DLBM	DVBM	dhbm?
	Byte(2-3) RMW	DLBM	DLBM	DHBM	DVBM	dhbm?
D32:RMW, MD32:RMW	Quad-byte Read-Modify-Write transfers					
	Byte(0-3) RMW	DLBM	DLBM	DLBM	DVBM	DLBM
D32:UAT	Unaligned transfers					
	Byte(0-2) Read or Write	DLBM	dhbm?	DLBM	DVBM	DLBM
	Byte(1-3) Read or Write	dhbm?	DLBM	DLBM	DVBM	DLBM
	Byte(1-2) Read or Write	DLBM	DLBM	DHBM	DVBM	DLBM

Table 25 (*continued*)

NOTE 1 During single-byte block transfers, the two data strobes are alternately driven low. Either data strobe might be driven low on the first transfer. If the first accessed byte location is Byte(0) or Byte(2), then the Master drives DS1* low first. If the first accessed byte location is Byte(1) or Byte(3), then it drives DS0* low first. A1 is valid only on the first data transfer (that is, until the Slave drives DTACK* or BERR* low the first time) and might be either high or low depending upon which byte the single byte block transfer begins with. If the first byte location is Byte(0) or Byte(1), then the Master drives A1 to low. If the first byte location is Byte(2) or Byte(3), then the Master drives A1 to high.

An example of the use of DS0*, DS1*, A1, A2, and LWORD* during a single-byte block transfer cycle that starts with Byte(2) is given below.

		DS1*	DS0*	A1	A2	LWORD*
First data transfer	Byte(2)	DLBM	DHBM	DHBM	DVBM	dhbm?
	Byte(3)	DHBM	DLBM	dxbm?	DVBM	dxbm?
	Byte(0)	DLBM	DHBM	dxbm?	DVBM	dxbm?
	Byte(1)	DHBM	DLBM	dxbm?	DVBM	dxbm?
Last data transfer	Byte(2)	DLBM	DHBM	dxbm?	DVBM	dxbm?

NOTE 2 During a double-byte block transfer, A1 is valid only on the first data transfer (that is, until the Slave drives DTACK* or BERR* low the first time) and is driven either high or low depending upon what double-byte group the double-byte block transfer begins with. If the first double-byte group is Byte(0-1), then the Master drives A1 low. If the first double-byte group is Byte(2-3), then the Master drives A1 high.

Table 26 – Use of the data lines to transfer data

Mnemonic	Type of cycles	A[31..1] LWORD*	D[31..24]	D[23..16]	D[15..8]	D[7..0]
ADO	Address-Only	dxbm?[2]	dxbm?	dxbm?	dxbm?	dxbm?
ADOH	Address-Only With-Handshake	dxbm?[2]	dxbm?	dxbm?	dxbm?	dxbm?
D08(EO)	Single-even-byte transfers					
	Byte(0) Read	dxbm?[2]	dxbs?	dxbs?	DVBS	dxbs?
	Byte(2) Read	dxbm?[2]	dxbs?	dxbs?	DVBS	dxbs?
	Byte(0) Write	dxbm?[2]	dxbm?	dxbm?	DVBM	dxbm?
	Byte(2) Write	dxbm?[2]	dxbm?	dxbm?	DVBM	dxbm?
D08(EO) or D08(O)	Single-odd-byte transfers					
	Byte(1) Read	dxbm?[2]	dxbs?	dxbs?	dxbs?	DVBS
	Byte(3) Read	dxbm?[2]	dxbs?	dxbs?	dxbs?	DVBS
	Byte(1) Write	dxbm?[2]	dxbm?	dxbm?	dxbm?	DVBM
	Byte(3) Write	dxbm?[2]	dxbm?	dxbm?	dxbm?	DVBM
D16	Double-byte transfers					
	Byte(0-1) Read	dxbm?[2]	dxbs?	dxbs?	DVBS	DVBS
	Byte(2-3) Read	dxbm?[2]	dxbs?	dxbs?	DVBS	DVBS
	Byte(0-1) Write	dxbm?[2]	dxbm?	dxbm?	DVBM	DVBM
	Byte(2-3) Write	dxbm?[2]	dxbm?	dxbm?	DVBM	DVBM
D32	Quad-byte transfers					
	Byte(0-3) Read	dxbm?[2]	DVBS	DVBS	DVBS	DVBS
	Byte(0-3) Write	dxbm?[2]	DVBM	DVBM	DVBM	DVBM
MD32	Multiplexed quad-byte transfers					
	Quad-Byte Read	DVBS[2]	dxbs?	dxbs?	DVBS	DVBS
	Quad-Byte Write	DVBM[2]	dxbm?	dxbm?	DVBM	DVBM

Table 26 (*continued*)

Mnemonic	Type of cycles	A[31..1] LWORD*	D[31..24]	D[23..16]	D[15..8]	D[7..0]
D08(EO): BLT	Single-byte block transfers					
	Single-Byte Block Read	dxbm?[2]	dxbs?	dxbs?	Note 1	Note 1
	Single-Byte Block Write	dxbm?[2]	dxbm?	dxbm?	Note 1	Note 1
D16:BLT	Double-byte block transfers					
	Double-Byte Block Read	dxbm?[2]	dxbs?	dxbs?	DVBS	DVBS
	Double-Byte Block Write	dxbm?[2]	dxbm?	dxbm?	DVBM	DVBM
D32:BLT	Quad-byte block transfers					
	Quad-Byte Block Read	dxbm?[2]	DVBS	DVBS	DVBS	DVBS
	Quad-Byte Block Write	dxbm?[2]	DVBM	DVBM	DVBM	DVBM
MD32:BLT	Quad-byte multiplexed block transfers					
	Quad-Byte Block Read	DVBS[2]	dxbs?	dxbs?	DVBS	DVBS
	Quad-Byte Block Write	DVBM[2]	dxbm?	dxbm?	DVBM	DVBM
MBLT	Eight-byte block transfers					
	Eight-Byte Block Read	DVBS	DVBS	DVBS	DVBS	DVBS
	Eight-Byte Block Write	DVBM	DVBM	DVBM	DVBM	DVBM
D08(EO): RMW	Single-byte Read-Modify-Write transfers					
	Byte(0) RMW	dxbm?[2]	dxbb?	dxbb?	DVBB	dxbb?
	Byte(1) RMW	dxbm?[2]	dxbb?	dxbb?	dxbb?	DVBB
	Byte(2) RMW	dxbm?[2]	dxbb?	dxbb?	DVBB	dxbb?
	Byte(3) RMW	dxbm?[2]	dxbb?	dxbb?	dxbb?	DVBB
D16:RMW	Double-byte Read-Modify-Write transfers					
	Byte(0-1) RMW	dxbm?[2]	dxbb?	dxbb?	DVBB	DVBB
	Byte(2-3) RMW	dxbm?[2]	dxbb?	dxbb?	DVBB	DVBB
D32: RMW	Quad-byte Read-Modify-Write transfers					
	Byte(0-3) RMW	dxbm?[2]	DVBB	DVBB	DVBB	DVBB
MD32: RMW	Multiplexed quad-byte Read-Modify-Write transfers					
	Quad-Byte RMW	DVBB[2]	dxbb?	dxbb?	DVBB	DVBB
D32:UAT	Unaligned transfers					
	Byte(0-2) Read	dxbm?[2]	DVBS	DVBS	DVBS	dxbs?
	Byte(1-3) Read	dxbm?[2]	dxbs?	DVBS	DVBS	DVBS
	Byte(1-2) Read	dxbm?[2]	dxbs?	DVBS	DVBS	dxbs?
	Byte(0-2) Write	dxbm?[2]	DVBM	DVBM	DVBM	dxbm?
	Byte(1-3) Write	dxbm?[2]	dxbm?	DVBM	DVBM	DVBM
	Byte(1-2) Write	dxbm?[2]	dxbm?	DVBM	DVBM	dxbm?

Table 26 (*continued*)

NOTE 1 During single-byte block transfers, data is transferred 8 bits at a time over D07-D00 or D15-D08. A single-byte block read example is given below:

	D[15..8]	D[7..0]
First data transfer	DVBS dxbs? DVBS dxbs? DVBS dxbs?	dxbs? DVBS dxbs? DVBS dxbs? DVBS
Last data transfer	DVBS	dxbs?

NOTE 2 A[15..1] and LWORD* are used to carry data only during MBLT and MD32 cycles. In addition, A[31..16] are used to carry data only during MBLT cycles. A[31..16] may be driven by Slave during MD32 read cycles and by Master during MD32 write cycles; however, they carry no valid information.

Table 27 – Master, slave, and location monitor timing parameters

Timing parameter number	Master (See also Table 29)		Slave (See also Table 30)		Location monitor (See also Table 31)	
	MIN	MAX	MIN	MAX	MIN	MAX
1	0					
2	0					
3	60					
4	35		10		10	
5	40		30		30	
6	0		0			
7	0		0			
8	35		10		10	
9	0		0			
10	0		–10		–10	
11	40		30		30	
12	35		10		10	
13		10		20		20
14	0		0			
15	0		0			
16	0		0			
17	40		30		30	
18	0		0			
18A	0		0			
19	40		30		30	
20	0		0			
21	0		0			
22	0		0			
23	10		0		0	
24A	0					
24B	0					
25		25				
26	0		0			
27	–25		0			
28	30	2T	30			
28A			30			
28B			0			
29	0		0			
30	0		0			
30A			0	30		
31	0		0			
32			10		10	
33			30		30	
44	30		30			
44A			30			
45	225	2T	225			
46	0		0	30		
46A			0	30		
47	0	200	0	225		
48	0		0			

NOTE 1 All times are in nanoseconds.

NOTE 2 T = time-out value in microseconds.

Table 28 – Bus-timer timing parameters (See also Table 32)

Parameter number	Bus timer	
	MIN	MAX
28	T	2T
30	0	
NOTE T = time-out value in microseconds.		

Table 29 – Master, timing RULEs and OBSERVATIONs

The numbers correspond to the timing parameters specified in Table 27.	
1	**RULE 2.27** When taking control of the DTB, the Master MUST NOT drive any of the IACK*, AM[5..0], A[31..1], LWORD*, D[31..0], WRITE*, DS0*, DS1*, or AS* lines until after the previous Master or interrupt handler allows AS* to rise above the low level. **OBSERVATION 2.35** Clause 3 describes how a Master's requester is granted use of the DTB.
2	**RULE 2.28** When taking control of the DTB, the Master MUST NOT drive any of the IACK*, AM[5..0], A[31..1], LWORD*, D[31..0], WRITE*, DS0*, DS1*, or AS* lines until after its requester is granted the bus. **OBSERVATION 2.36** Clause 3 describes how a Master's requester is granted use of the DTB.
3	**RULE 2.29** When taking control of the DTB, the Master MUST NOT drive AS* low until this time after the previous Master or interrupt handler allows AS* to rise above the low level. **OBSERVATION 2.37** Timing parameter 3 ensures that timing parameter 5 for Slaves is guaranteed when there is an interchange of the DTB Mastership.
4	**RULE 2.30** The Master **MUST NOT** drive AS* low until IACK* has been high, and the required lines among A[31..1], AM[5..0], and LWORD* have been valid for this minimum time. **OBSERVATION 2.38** Table 19 specifies the lines among A[31..1] that the Master is required to drive. Table 4 specifies how it uses AM[5..0], and Table 25 how it uses LWORD*.
5	**RULE 2.31** When using the DTB for two consecutive cycles, the Master **MUST NOT** drive AS* low until it has been high for this minimum time.
6	**RULE 2.32** After all read cycles, the Master **MUST NOT** drive any of D[31..0] until both DTACK* and BERR* are high.

Table 29 (*continued*)

7	**RULE 2.33a** During all read cycles except during A64, A40 and MBLT, and except during the A40BLT address phase, the Master **MUST NOT** drive DSA* low until it has released all of the D[31..0] lines. In addition, during MBLT and MD32 read cycles and when a data transfer follows A64 and A40 addressing, the Master **MUST NOT** drive DSA* low until it has released all the D[31..0], A[31..1], and LWORD* lines.
8	**RULE 2.34a** During write cycles and when using A64 or A40 addressing, the Master **MUST NOT** drive DSA* low until the required lines among D[31..0] have been valid for this minimum time. In addition, during MBLT, A40BLT, and MD32 write cycles, the Master **MUST NOT** drive DSA* low until A[31..1] and LWORD* have been valid for this minimum time. **OBSERVATION 2.39a** Table 26 specifies the lines among D[31..0] that a Master is required to drive. Table 24 specifies how A64 Master and A40 Master use D[31..0] for A64 and A40 addressing.
9	**RULE 2.35** The Master **MUST NOT** drive DSA* low until both DTACK* and BERR* are high.
10	**RULE 2.36** The Master **MUST NOT** drive DSA* low until it has driven AS* low.
11	**RULE 2.37** The Master **MUST NOT** drive DSA* low until DS0* and DS1* have both been simultaneously high for this minimum time.
12	**RULE 2.38** The Master **MUST NOT** drive DSA* low for the first time until WRITE* has been valid for this minimum time.
13	**RULE 2.39** During cycles where the Master drives both DS0* and DS1* low, it **MUST** drive DSB* low within this maximum time after it drives DSA* low. **OBSERVATION 2.40** Timing parameter 13 does not apply to transfers where either DS0* is driven low or DS1* is driven low, but not both.
14	**RULE 2.40** During all data transfer cycles, excluding A16, A24 and A32 Read-Modify-Write and Retry cycles, the Master **MUST** drive A[31..1] and LWORD* with valid addressing information until it detects the first falling edge on DTACK* or BERR*. **OBSERVATION 2.41** During all data transfer cycles, except block transfer cycles and all Read-Modify-Write cycles, there will be only one falling edge on DTACK* or BERR*.
15	**RULE 2.41** During all Read-Modify-Write cycles, the Master **MUST** hold A[31..1] valid and maintain the appropriate level on LWORD* until it detects the second falling edge on DTACK* or BERR*.

Table 29 (*continued*)

16	**RULE 2.42** During all data transfer cycles, the Master **MUST** maintain a valid AM code, and ensure that IACK* stays high until it detects the last falling edge on DTACK* or BERR*. **OBSERVATION 2.42** During all data transfer cycles, except block transfer and Read-Modify-Write cycles, there will be only one falling edge on DTACK* or BERR*.
17	**RULE 2.43** The Master **MUST NOT** change the levels on IACK*, A[31..1], AM[5..0], or LWORD* for this minimum time after it drives AS* low.
18	**RULE 2.44a** During all data transfer cycles except MBLT, A40BLT, and MD32 read cycles, the Master **MUST** hold AS* low until it detects the last falling edge on DTACK* or BERR*.
18A	**RULE 2.94** During MBLT, A40BLT, and MD32 read cycles, the Master **MUST** hold AS* low until it detects the last rising edge on DTACK* or BERR*.
19	**RULE 2.45** The Master **MUST** hold AS* low for this minimum time.
20	**RULE 2.46a** Once a Master has driven DSA* low, it **MUST** maintain that line low until it detects the falling edge on DTACK*, RETRY*, or BERR*. **OBSERVATION 2.102** During a read cycle, data may not be valid at the bus Master for up to 25 ns after DTACK* is detected low. (See parameter 27.) To ensure that valid data is read, the bus Master should maintain DSA* asserted for at least 25 ns after detecting DTACK* low.
21	**RULE 2.47a** Once a Master has driven DSB* low, it **MUST** maintain that line low until it detects the falling edge on DTACK*, RETRY*, or BERR*. See OBSERVATION 2.102.
22	**RULE 2.48a** During all write cycles and during A64 or A40 addressing, once the Master has driven DSA* low, it **MUST NOT** change any of D[31..0] until it detects DTACK*, BERR* or RETRY* low. In addition, during MBLT and MD32 write data transfers, once the Master has driven DSA* low, it **MUST NOT** change any of A[31..1] and LWORD* until it detects DTACK*, BERR* or RETRY* low.
23	**RULE 2.49** Once a Master has driven DSA* low, it **MUST NOT** change the level of WRITE* until this minimum time after DSB* is high.
24A	**RULE 2.50** **IF** the Master drives or releases AS* high after its requester releases BBSY*, **THEN** it **MUST** release IACK*, AM[5..0], A[31..1], LWORD*, D[31..0], WRITE*, DS0*, and DS1* before allowing AS* to rise above the low level. **OBSERVATION 2.43** Subclause 3.2 describes how a Master's requester releases the BBSY* line.

Table 29 (*continued*)

24B	**RULE 2.51** **IF** the Master drives or releases AS* high before its requester releases BBSY*, **THEN** it **MUST** release AS*, IACK*, AM[5..0], A[31..1], LWORD*, D[31..0], WRITE*, DS0*, and DS1* before allowing its requester to release BBSY*. **OBSERVATION 2.44** Subclause 3.2 describes how a Master's requester releases the BBSY* line.
25	**RULE 2.52** When the Master drives or releases AS* high after its requester releases BBSY*, then it **MUST** release AS* within this time after allowing it to rise above the low level. **OBSERVATION 2.45** Subclause 3.2 describes how a Master's requester releases the BBSY* line.
26	**OBSERVATION 2.46a** During all read cycles except A64, A40, MBLT and A40BLT address phase, the Master is guaranteed that the responding Slave will not drive D[31..0] until the Master drives DSA* low. In addition, during MBLT and A40BLT read cycles and all A64 and A40 read cycles, the Master is guaranteed that the responding Slave will not drive any of D[31..0], A[31..1], and LWORD* lines until the Master drives DSA* low for the second time.
27	**OBSERVATION 2.47a** During read cycles, the Master is guaranteed that D[31..0] will be valid within 25 ns after DTACK* goes low. In addition, during MBLT, A40BLT, and MD32 read cycles, the Master is guaranteed that A[31..1] and LWORD* will be valid within 25 ns after DTACK* goes low. This time does not apply to cycles where the Slave drives BERR* low instead of DTACK*.
28	**OBSERVATION 2.48** The Master is guaranteed that neither DTACK* nor BERR* will go low until this minimum time after it drives DSA* low. The bus timer guarantees the Master that if DTACK* is not detected low after its time-out period has elapsed, and within twice its time-out period, then the bus timer will drive BERR* low.
29	**OBSERVATION 2.49a** During read cycles, the Master is guaranteed that D[31..0] remain valid until it drives DSA* high. In addition, during MBLT, A40BLT, and MD32 read cycles, the Master is guaranteed that A[31..1] and LWORD* remain valid until it drives DSA* high.
30	**OBSERVATION 2.50** The Master is guaranteed that DTACK*, and BERR* will not go high until it drives both DS0* and DS1* high.
31	**OBSERVATION 2.51a** During all read cycles, the Master is guaranteed that D[31..0] lines are released by the time it detects the rising edge of DTACK*, RETRY* or BERR*. In addition, during MBLT, A40BLT, and MD32 read cycles, the Master is guaranteed that A[31..1] and LWORD* lines have been released by the time it detects the rising edge of DTACK*, BERR* or RETRY*.

Table 29 (*continued*)

44	**OBSERVATION 2.103** The Master is guaranteed that RETRY* will not go low until this minimum time after it drives DSA* low.
45	**OBSERVATION 2.104** **IF** the Slave drives RETRY* low, **THEN** the Master is guaranteed that neither DTACK* nor BERR* will be driven low until this minimum time after RETRY* has been driven low.
46	**OBSERVATION 2.105** The Master is guaranteed that RETRY* will not go high until it drives both DS0* and DS1* high.
47	**RULE 2.95** **IF** the Master responds to the falling edge on RETRY*, **THEN** it **MUST** drive the data strobes high within this specified time after detecting the falling edge on RETRY*.
48	**RULE 2.96** **IF** the Master responds to RETRY* by driving data strobes high within the time specified by Timing Parameter 47 after detecting the falling edge on RETRY*, **THEN** it **MUST NOT** drive or release AS* high and **MUST NOT** change the levels on IACK* and AM[5..0] until it detects the rising edge on RETRY*. In addition, during address phase, it **MUST NOT** change the levels on LWORD* and the address lines, including D[31..0] for A40 and A64 addressing, until it detects the rising edge on RETRY*. **OBSERVATION 2.106** Since RETRY* may not be monitored by all Masters and Slaves in the system, the Master responding to RETRY* must guarantee that the data lines have been released prior to the transfer of bus mastership. This is guaranteed by timing parameters 31 and 48.

Table 30 – Slave, timing RULEs and OBSERVATIONs

The numbers correspond to the timing parameters specified in Table 27.	
4	**OBSERVATION 2.52** All Slaves are guaranteed that the IACK*, A[31..1], AM[5..0], and LWORD* lines have been valid for this minimum time when they detect a falling edge on AS*.
5	**OBSERVATION 2.53** All Slaves are guaranteed this minimum high time on AS* between DTB cycles.
6	**OBSERVATION 2.54** During read cycles, the responding Slave is guaranteed that none of the D[31..0] lines will be driven by any other module until it releases DTACK* and BERR* to high.
7	**OBSERVATION 2.55a** During all read cycles except A64, A40, MBLT and A40BLT address phase, the responding Slave is guaranteed that the D[31..0] lines are released by the time DSA* goes low. In addition, during MBLT and A40BLT read cycles and when a data transfer follows A64 and A40 addressing, the responding Slave is guaranteed that the D[31..0], A[31..1], and LWORD* lines are released by the time DSA* goes low.
8	**OBSERVATION 2.56a** During write cycles and during A64 or A40 addressing, all Slaves are guaranteed that the D[31..0] lines have been valid for this minimum time when they detect a falling edge on DSA*. In addition, during MBLT and MD32 write cycles, all Slaves are guaranteed that the A[31..1] and LWORD* lines have been valid for this minimum time when they detect a falling edge on DSA*.
9	**OBSERVATION 2.57** The responding Slave is guaranteed that neither DS0* nor DS1* will go low until it drives DTACK* and BERR* high.
10	**OBSERVATION 2.58** Due to bus skew, Slaves on the DTB might detect a falling edge on DSA* before detecting the falling edge on AS*. However, Slaves are guaranteed that a falling edge on DSA* will not precede the falling edge on AS* by more than this time.
11	**OBSERVATION 2.59** Slaves are guaranteed this minimum time during which both DS0* and DS1* are simultaneously high.
12	**OBSERVATION 2.60** Slaves are guaranteed that WRITE* has been valid for this minimum time before the first falling edge on DSA*.
13	**OBSERVATION 2.61** **IF** both data strobes are going to be driven low, **THEN** the responding Slave is guaranteed that DSB* will go low within this maximum time after DSA* has gone low.
14	**OBSERVATION 2.62** During all DTB cycles, excluding A16, A24 and A32 Read-Modify-Write cycles, the responding Slave is guaranteed that A[31..1] and LWORD* carry valid addressing information until it drives DTACK* or BERR* low for the first time, provided that it does so within the bus time-out period.

Table 30 (*continued*)

15	**OBSERVATION 2.63** During all Read-Modify-Write cycles, the responding Slave is guaranteed that the A[31..1] and LWORD* lines remain valid until it drives DTACK* or BERR* low for the second time, provided that it does so within the bus time-out period.
16	**OBSERVATION 2.64** The responding Slave is guaranteed that IACK* and AM[5..0] remain valid until it drives DTACK* or BERR* low for the last time, provided that it does so within the bus time-out period.
17	**OBSERVATION 2.65** Slaves are guaranteed that IACK*, A[31..1], AM[5..0], and LWORD* remain valid for this minimum time after the falling edge of AS*. During address-only cycles this time is guaranteed by the Master. During all other cycle types, this time is derived from timing parameters 10, 14, 16, and 28.
18	**OBSERVATION 2.66a** During all data transfer cycles except MBLT, A40BLT, and MD32 read cycles, the responding Slave is guaranteed that AS* will remain low until it drives DTACK* or BERR* low for the last time, provided that it does so within the bus time-out period.
18A	**OBSERVATION 2.107** During MBLT, A40BLT, and MD32 read cycles, the responding Slave is guaranteed that AS* will remain low until it drives DTACK* or BERR* high for the last time, provided that it does so within the bus time-out period.
19	**OBSERVATION 2.67** Slaves are guaranteed that AS* remains low for this minimum time.
20	**OBSERVATION 2.68a** The responding Slave is guaranteed that once DSA* goes low, it remains low until it drives DTACK*, RETRY*, or BERR* low, provided that the Slave does so within the bus time-out period.
21	**OBSERVATION 2.69a** The responding Slave is guaranteed that once DSB* goes low, it remains low until it drives DTACK*, RETRY*, or BERR* low, provided that the Slave does so within the bus time-out period.
22	**OBSERVATION 2.70a** During all write cycles and during A64 or A40 addressing, the responding Slave is guaranteed that D[31..0] remain valid until it drives DTACK* or BERR* low, provided that it does so within the bus time-out period. In addition, during MBLT and MD32 write data transfers, the responding Slave is guaranteed that A[31..1] and LWORD* remain valid until it drives DTACK* or BERR* low, provided that it does so within the bus time-out period.
23	**OBSERVATION 2.71** The responding Slave is guaranteed that WRITE* remains valid until both DS0* and DS1* are high.
26	**RULE 2.53a** During all read cycles except during A64, A40, MBLT, or A40BLT address phase, the responding Slave MUST NOT drive the D[31..0] lines until DSA* goes low. In addition, during MBLT and A40BLT read cycles and all A64 and A40 read cycles, the responding Slave MUST NOT drive any of D[31..0], A[31..1], and LWORD* lines until DSA* goes low for the second time.

Table 30 (*continued*)

27	**RULE 2.54a** During all read cycles, the responding Slave MUST NOT drive DTACK* low before it drives D[31..0] with valid data, except as noted in Timing Parameter 26. In addition, during MBLT and A40BLT read cycles, the responding Slave MUST NOT drive DTACK* low before it also drives A[31..1] and LWORD* with valid data. **OBSERVATION 2.72** RULE 2.54a does not apply to cycles where the responding Slave drives BERR* low instead of DTACK*.
28	**RULE 2.55** The responding Slave MUST wait this minimum time after DSA* goes low before driving DTACK* or BERR* low. **OBSERVATION 2.108** All Slaves are guaranteed that those bus lines that carry data during write cycles remain valid for this minimum time after DSA* goes low. This time is guaranteed by the requirement that the Master maintain data valid until it receives DTACK* or BERR* low.
28A	**RULE 2.97** Responding Slaves MUST wait this minimum time after detecting the first falling edge of DSA* before driving DTACK* to any level.
28B	**PERMISSION 2.19** Except for the first DSA*, responding Slaves MAY drive DTACK* high immediately after detecting the falling edge of DSA*.
29	**RULE 2.56a** During all read cycles, once the responding Slave has driven DTACK* low, it MUST NOT change D[31..0] until DSA* goes high. In addition, during MBLT, A40BLT, and MD32 read cycles, once the responding Slave has driven DTACK* low, it MUST NOT change A[31..1] or LWORD* until DSA* goes high.
30	**RULE 2.57** Once the responding Slave has driven DTACK* or BERR* low, it MUST NOT release them or drive DTACK* high until it detects both DS0* and DS1* high.
30A	**RULE 2.98** When a Slave drives DTACK* high at the end of a cycle, it MUST release DTACK* within this time.
31	**RULE 2.58a** During all read cycles, the responding Slave MUST release all of the D[31..0] lines before releasing DTACK* or BERR* high or driving RETRY* or DTACK* high. In addition, during MBLT, and MD32 read cycles, the responding Slave MUST release all of the A[31..1] and LWORD* lines before releasing DTACK* or BERR* high or driving RETRY* or DTACK* high.
32	**OBSERVATION 2.73** Slaves are guaranteed that IACK*, LWORD*, A[31..1], and AM[5..0] have been valid for this minimum time when they detect a falling edge on DSA*. This time is derived from timing parameters 4 and 10.
33	**OBSERVATION 2.74** During data transfer cycles, Slaves are guaranteed that either DS0*, or DS1*, or both remain low for at least this minimum time. This time is derived from timing parameter 28, where the responding Slave is required to wait a minimum time before driving BERR* or DTACK* low.

Table 30 (*continued*)

44	**RULE 2.99** The responding Slave MUST wait this minimum time after DSA* goes low before driving RETRY* low.
44A	**RULE 2.100** Responding Slaves **MUST** wait this minimum time after detecting the falling edge of DSA* before driving RETRY* to any level.
45	**RULE 2.101** Once the responding Slave drives RETRY* low, it **MUST NOT** drive either DTACK* or BERR* low until RETRY* has been low for this minimum time. **RULE 2.102** **IF** a Slave has asserted RETRY* low and the Master has not driven DSB* high within the time specified by parameter 47 **THEN** the SLAVE **MUST** complete the bus cycle with either DTACK* or BERR*.
46	**RULE 2.103** Once the responding Slave drives RETRY* low, it **MUST NOT** drive RETRY* high until it detects both DS0* and DS1* high. **RULE 2.104** Once the responding Slave drives RETRY* low, it **MUST** drive RETRY* high within this time after it detects both DS0* and DS1* high.
46A	**RULE 2.105** The Slave **MUST** release RETRY* within this time after it has driven RETRY* high at the end of the cycle.
47	**OBSERVATION 2.109** **IF** the Master has not driven DSB* high within this time after the Slave has driven RETRY* low, **THEN** the Slave is guaranteed that the Master will not respond to RETRY* and that DSA* will remain low until either DTACK* or BERR* is driven low.
48	**OBSERVATION 2.110** Slaves are guaranteed that AS* remains asserted and that IACK* and AM[5..0] remain valid until RETRY* is driven high. In addition, during address phase, Slaves are guaranteed that LWORD* and all address lines, including D[31..0] for A40 and A64 addressing, remain valid until RETRY* is driven high. This timing parameter guarantees that Slaves that do not monitor RETRY* will not see invalid addressing information while data strobes are low and neither DTACK* nor BERR* are low.

Table 31 – Location monitor, timing OBSERVATIONs

The numbers correspond to the timing parameters specified in Table 27.	
4	**OBSERVATION 2.75** Location Monitors are guaranteed that IACK*, A[31..1], AM[5..0], and LWORD* have been valid for this minimum time when they detect a falling edge on AS*.
5	**OBSERVATION 2.76** Location Monitors are guaranteed this minimum high time on AS* between DTB cycles.
8	**OBSERVATION 2.112** A64 Location Monitors are guaranteed that D[31..0] have been valid for this minimum time when they detect a falling edge on DSA*.
10	**OBSERVATION 2.77** Due to bus skew, Location Monitors might detect a falling edge on DSA* before detecting the falling edge on AS*. However, they are guaranteed that the falling edge on DSA* will not precede the falling edge on AS* by more than this time.
11	**OBSERVATION 2.78** Location Monitors are guaranteed this minimum time during which both data strobes are simultaneously high between consecutive data transfers.
12	**OBSERVATION 2.79** Location Monitors are guaranteed that WRITE* has been valid for this minimum time when they detect the first falling edge on DSA*.
13	**OBSERVATION 2.80** **IF** both data strobes are going to be driven low, **THEN** the Location Monitor is guaranteed that DSB* will go low within this maximum time after DSA* has gone low.
17	**OBSERVATION 2.81** The Location Monitor is guaranteed that IACK*, A[31..1], AM[5..0], and LWORD* will remain valid for this minimum time after the falling edge of AS*. During Address-Only cycles this time is guaranteed by the Master. During all other cycle types, this time is derived from timing parameters 10, 14, 16, and 28.
19	**OBSERVATION 2.82** The Location Monitor is guaranteed that AS* will remain low for this minimum time.
23	**OBSERVATION 2.83** The Location Monitor is guaranteed that the WRITE* line will remain valid until both data strobes go high.
32	**OBSERVATION 2.84** The Location Monitor is guaranteed that IACK*, LWORD*, A[31..1], and AM[5..0] have been valid for this minimum time when it detects a falling edge on DSA*.
33	**OBSERVATION 2.85** During data transfer cycles, the Location Monitor is guaranteed that DS0* and/or DS1* will remain low for at least this minimum time. This time is derived from timing parameter 28, where the responding Slave is required to wait a minimum time before driving BERR* or DTACK* to low.

Table 32 – BUS TIMER, timing RULEs

The numbers correspond to the timing parameters specified in Table 27.	
28	**RULE 2.59** The bus timer **MUST** wait at least its time-out time, but no longer than twice its time-out time, after the first data strobe goes low before driving BERR* low.
30	**RULE 2.60** Once it has driven BERR* low, the bus timer **MUST NOT** release BERR* until it detects both DS0* and DS1* high. **RECOMMENDATION 2.13** To ensure predictable system operation this time-out value should be longer than longest Slave response time in the system.

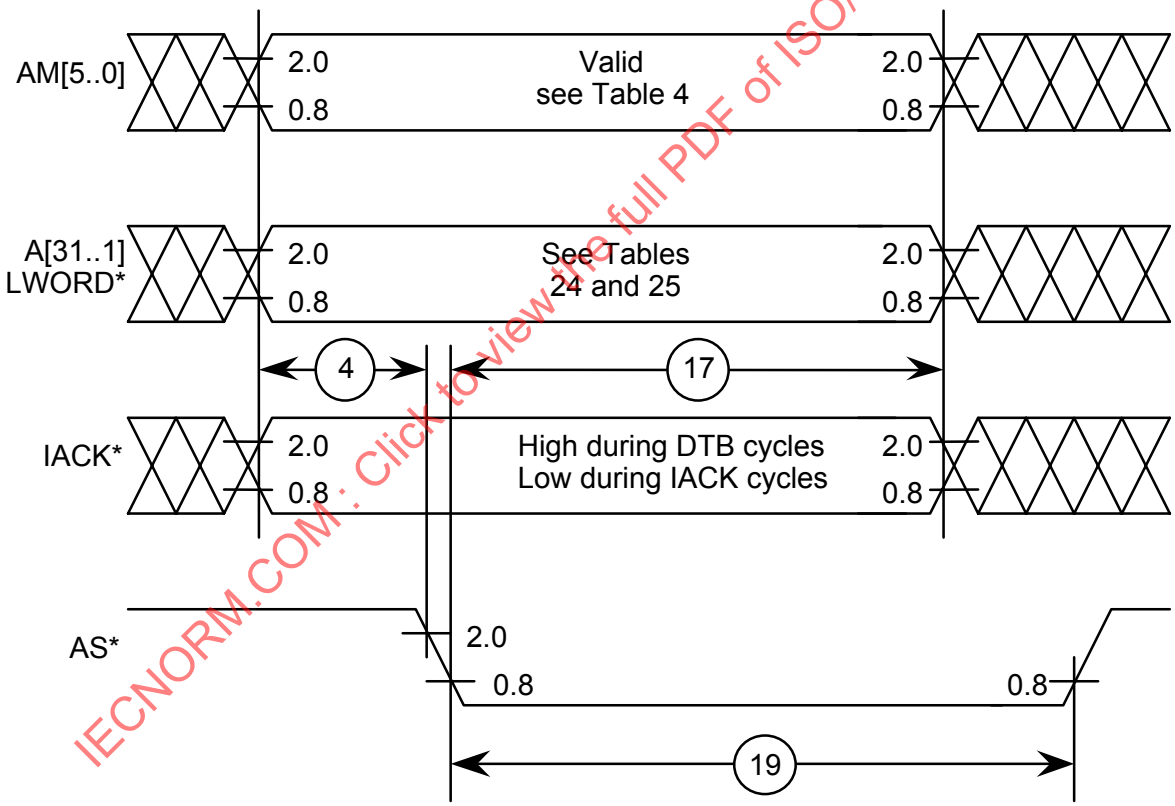
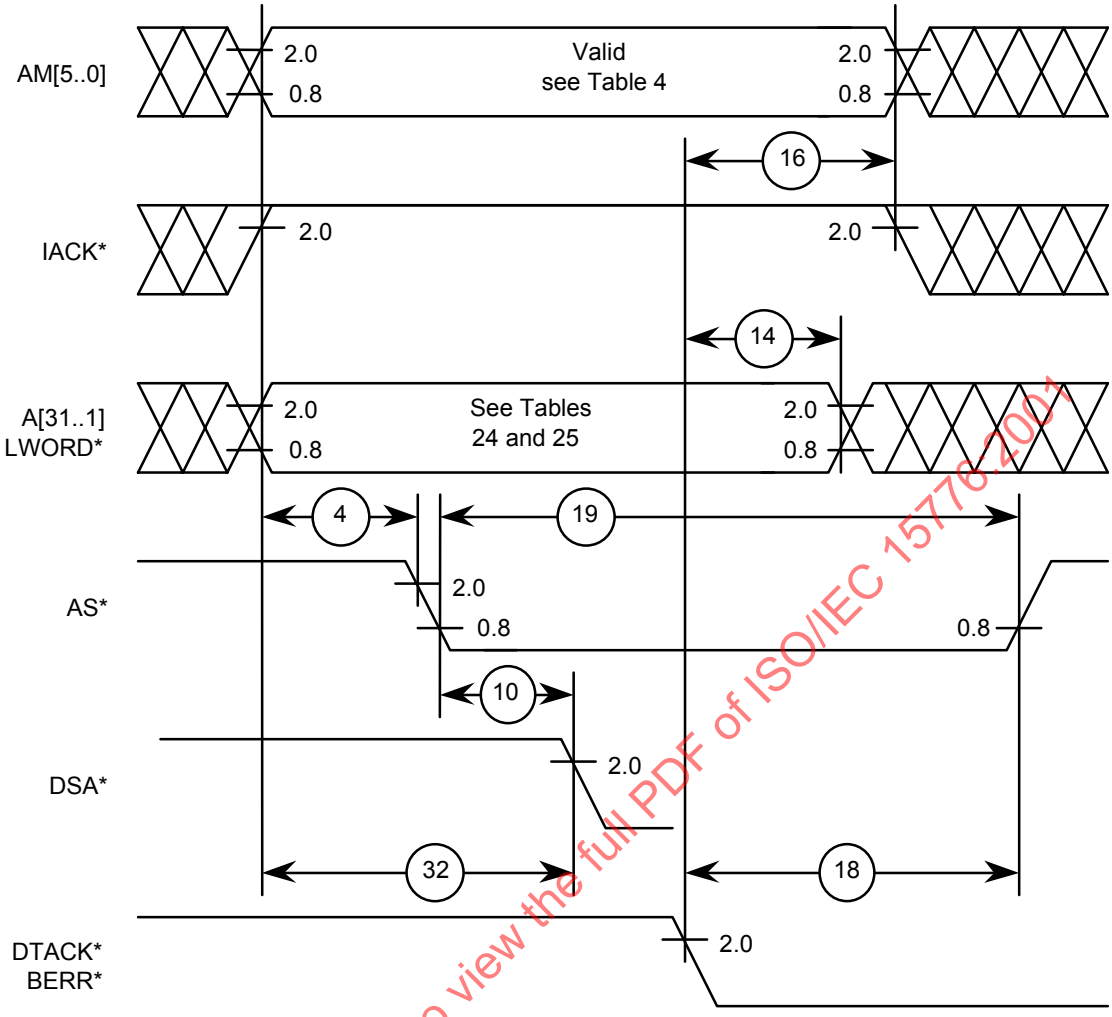
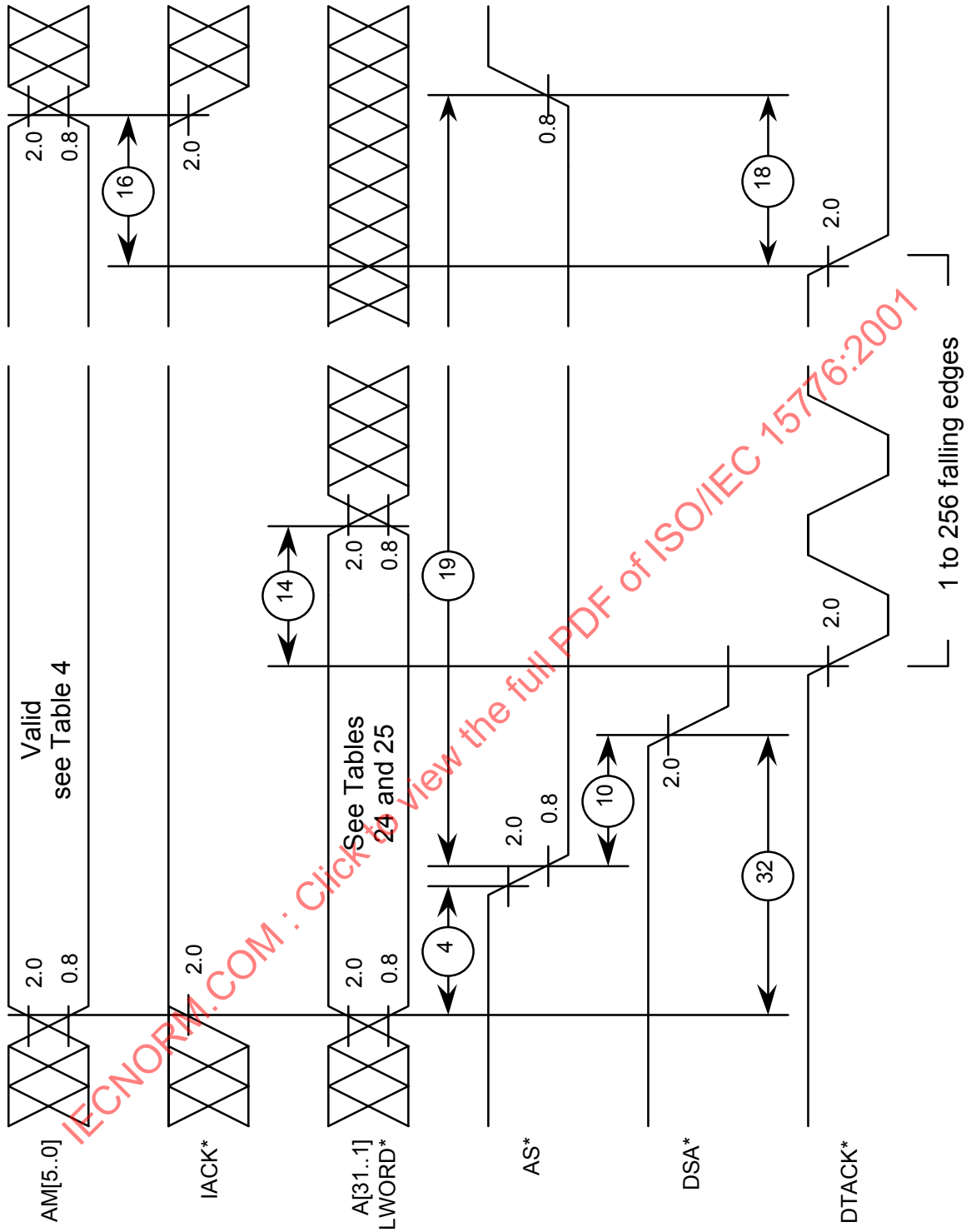


Figure 17 – Address broadcast timing – All cycles



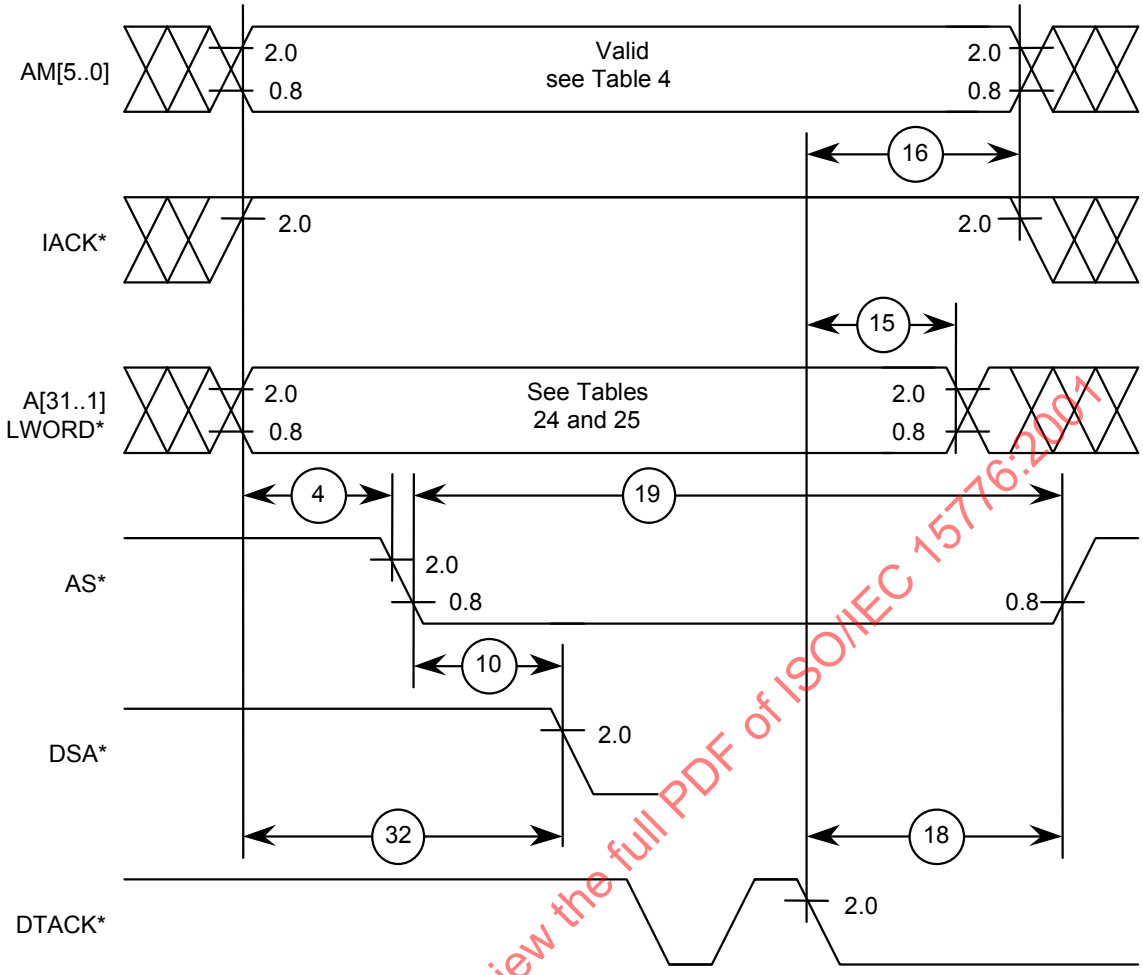
Address; Broadcast Timing
Single-Even-Byte Transfers
Single-Odd-Byte Transfers
Double-Byte Transfers
Quad-Byte Transfers
Unaligned Transfers

Figure 18 – A16, A24, A32 master, responding slave, and location monitor



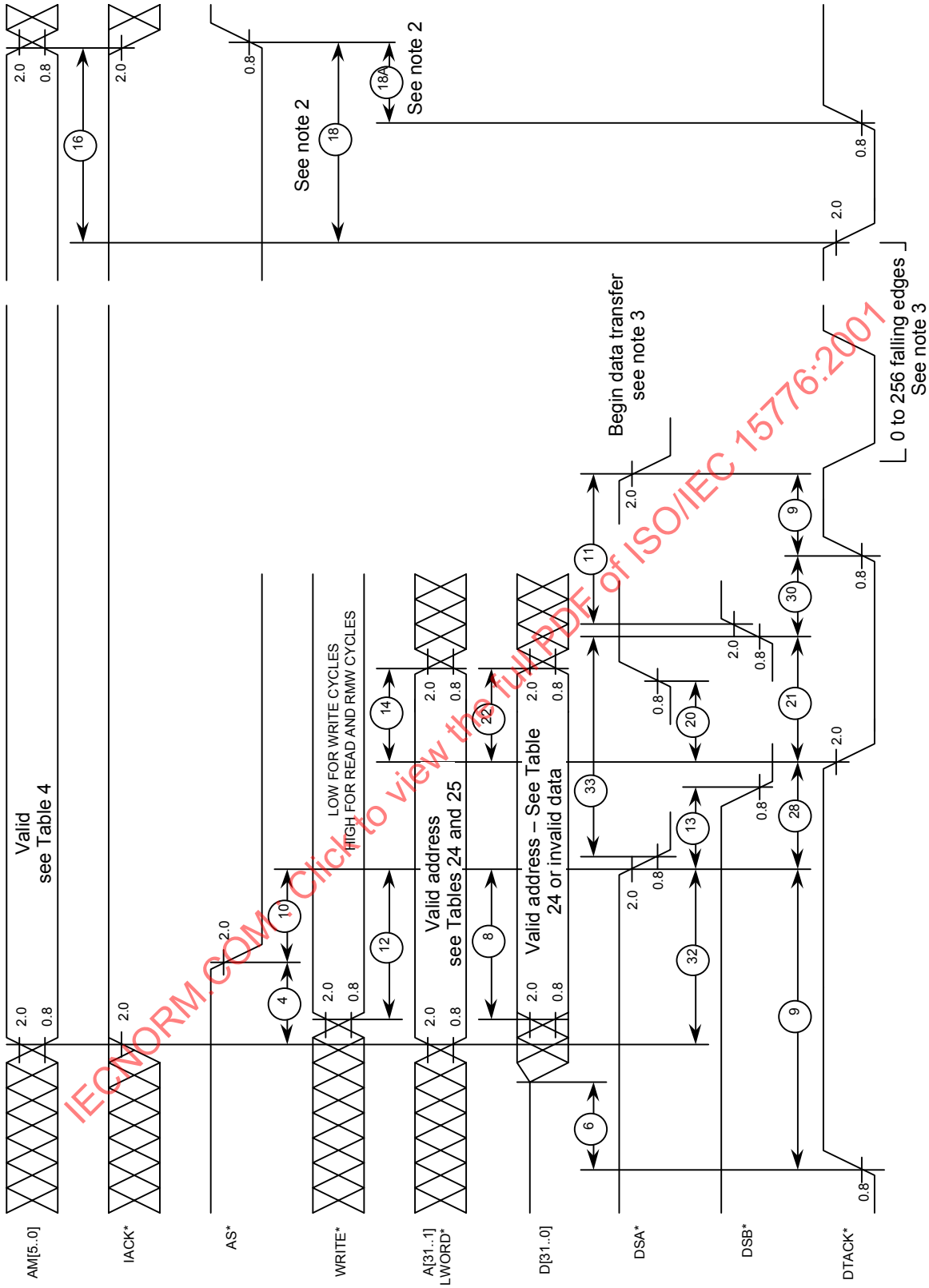
Single-Byte Block Transfers
Double-Byte Block Transfers
Quad-Byte Block Transfers

Figure 19 – Master, slave, and location monitor – A16, A24 and A32 address broadcast timing



Single-Byte RMW Cycles
Double-Byte RMW Cycles
Quad-Byte RMW Cycles

Figure 20 – Master, slave, and location monitor A16, A24, and A32 address broadcast timing



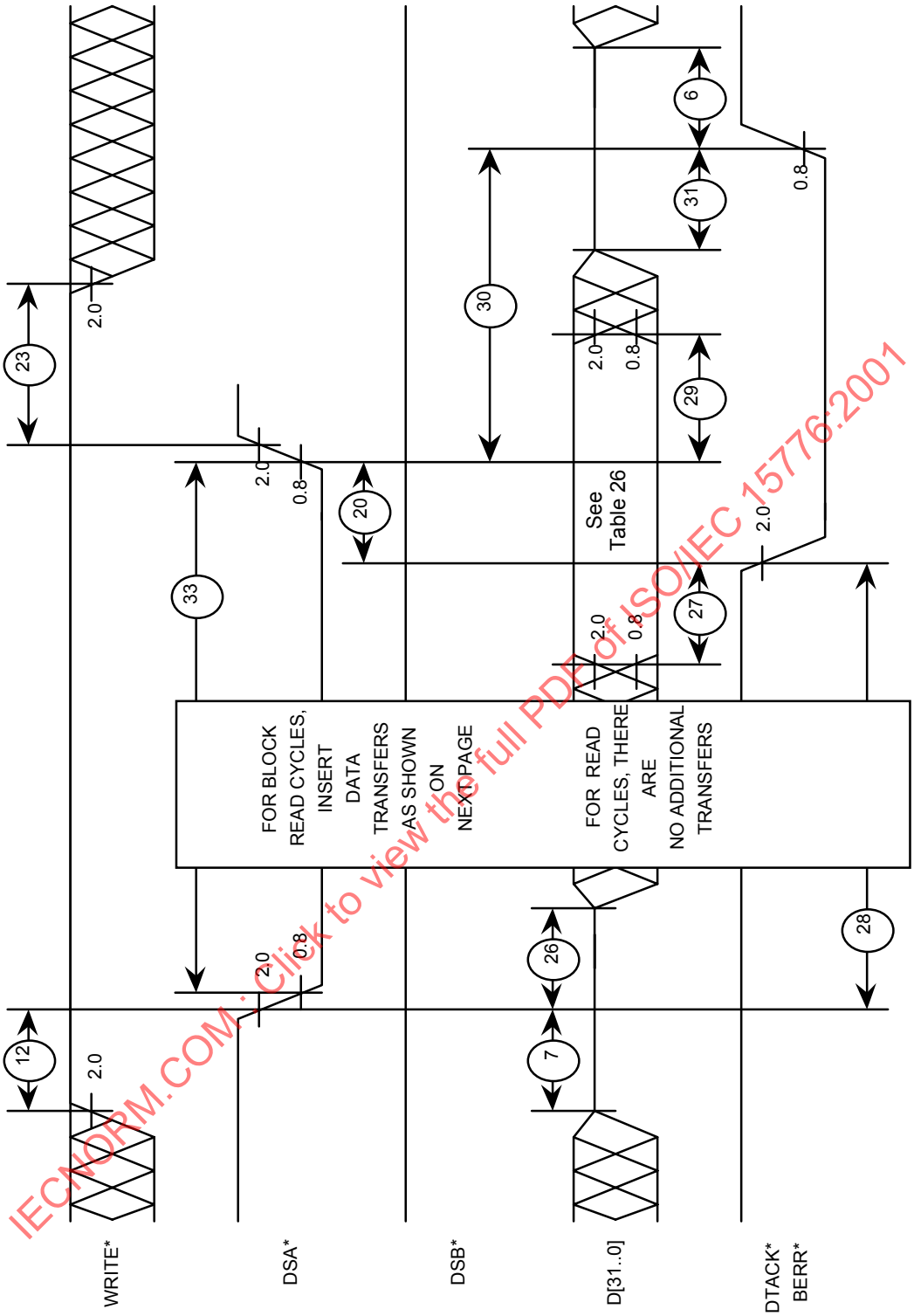
NOTE 1 The rising and falling edges of DSA* and DSB* are shown disconnected to indicate that the signal that is the first to fall is not necessarily the one that is the first to rise.

NOTE 2 During MBLT and MD32 read cycles, apply timing parameter 18A. During all other cycles, apply timing parameter 18.

NOTE 3 All ADOH cycles terminate after the first DTACK* without data transfers.

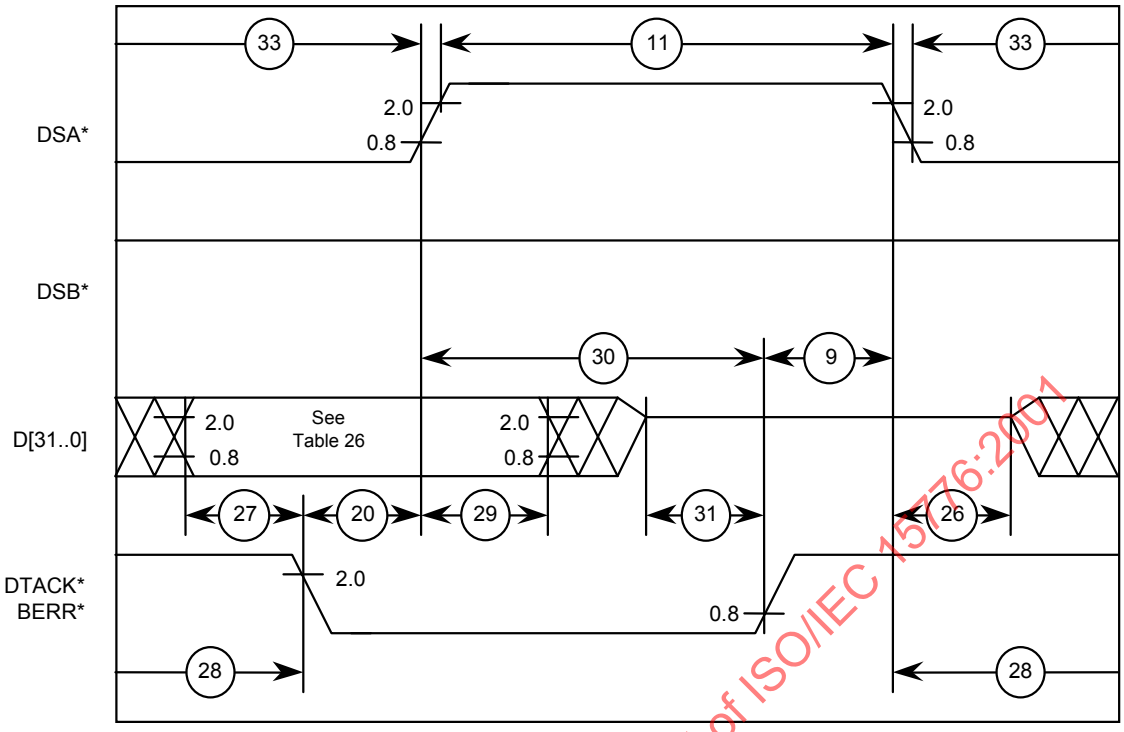
All data transfer cycles.

Figure 21 – Master, slave and location monitor – A64, A40, and ADOH address broadcast timing



Single Byte Read
Byte(0-2) UAT Read
Byte(1-3) UAT Read
Single-Byte Block Read

Figure 22 – Master, slave and location monitor data transfer timing



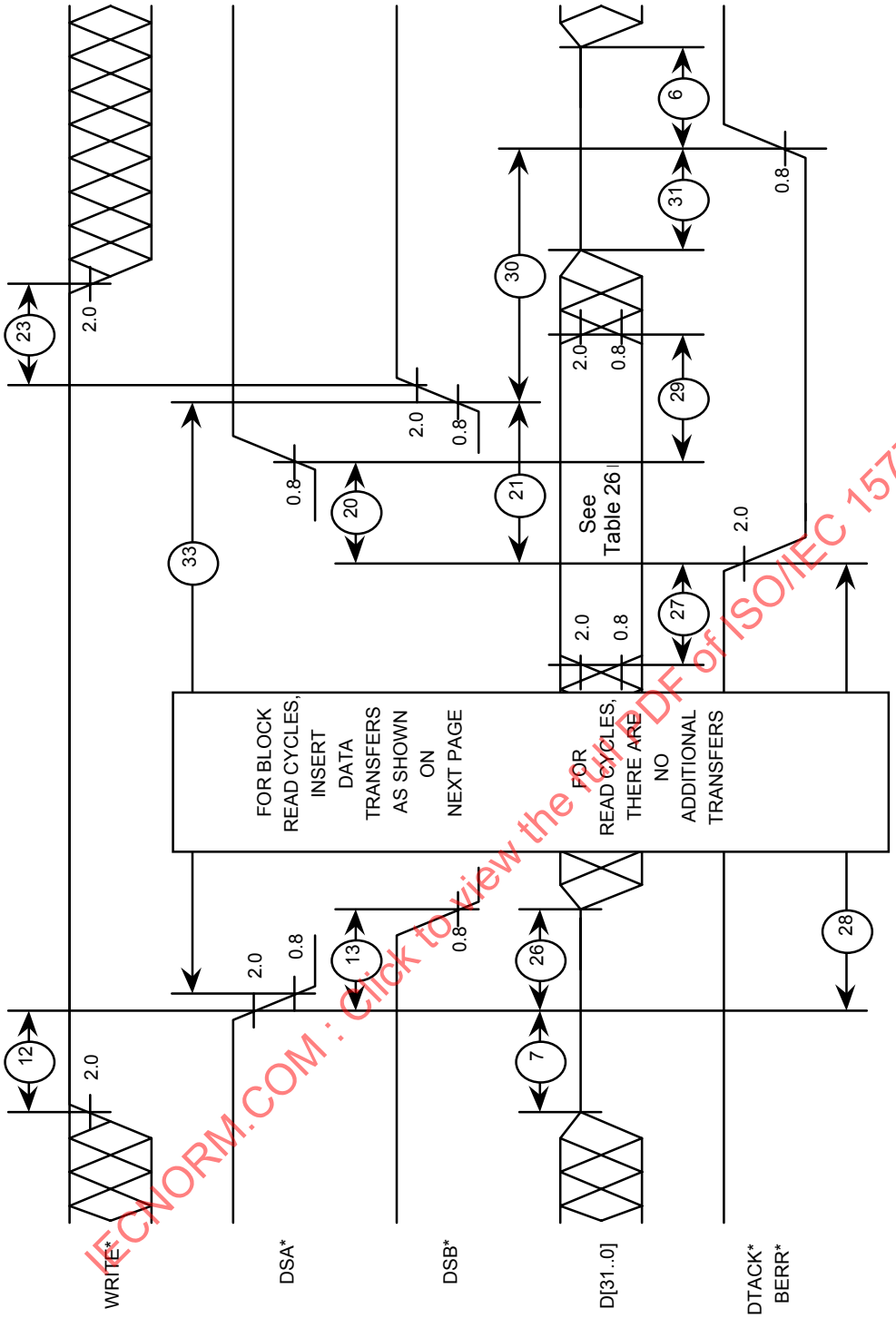
Single Byte Read

Byte(0-2) UAT Read

Byte(1-3) UAT Read

Single-Byte Block Read

Figure 22 – Master, slave and location monitor data transfer timing (*continued*)



NOTE The falling and rising edges of DSA* and DSB* are not connected to indicate that the signal that is the first to fall is not necessarily the one that is the first to rise.

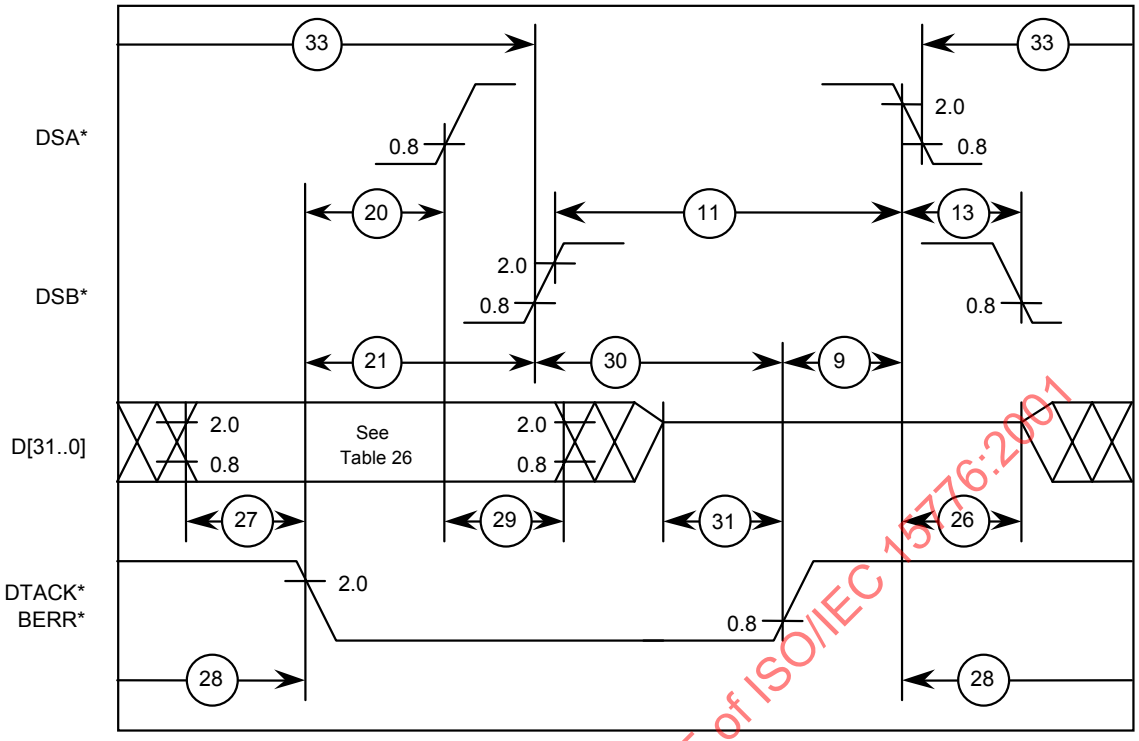
Byte(0-1) Read, Byte(2-3) Read

Byte(0-3) Read, Byte(1-2) Read

Double-Byte Block Read

Quad-Byte Block Read

Figure 23 – Master, slave and location monitor data transfer timing



NOTE The rising and falling edges of DSA* and DSB* are not connected to indicate that the signal that is the first to rise is not necessarily the one that is the first to fall.

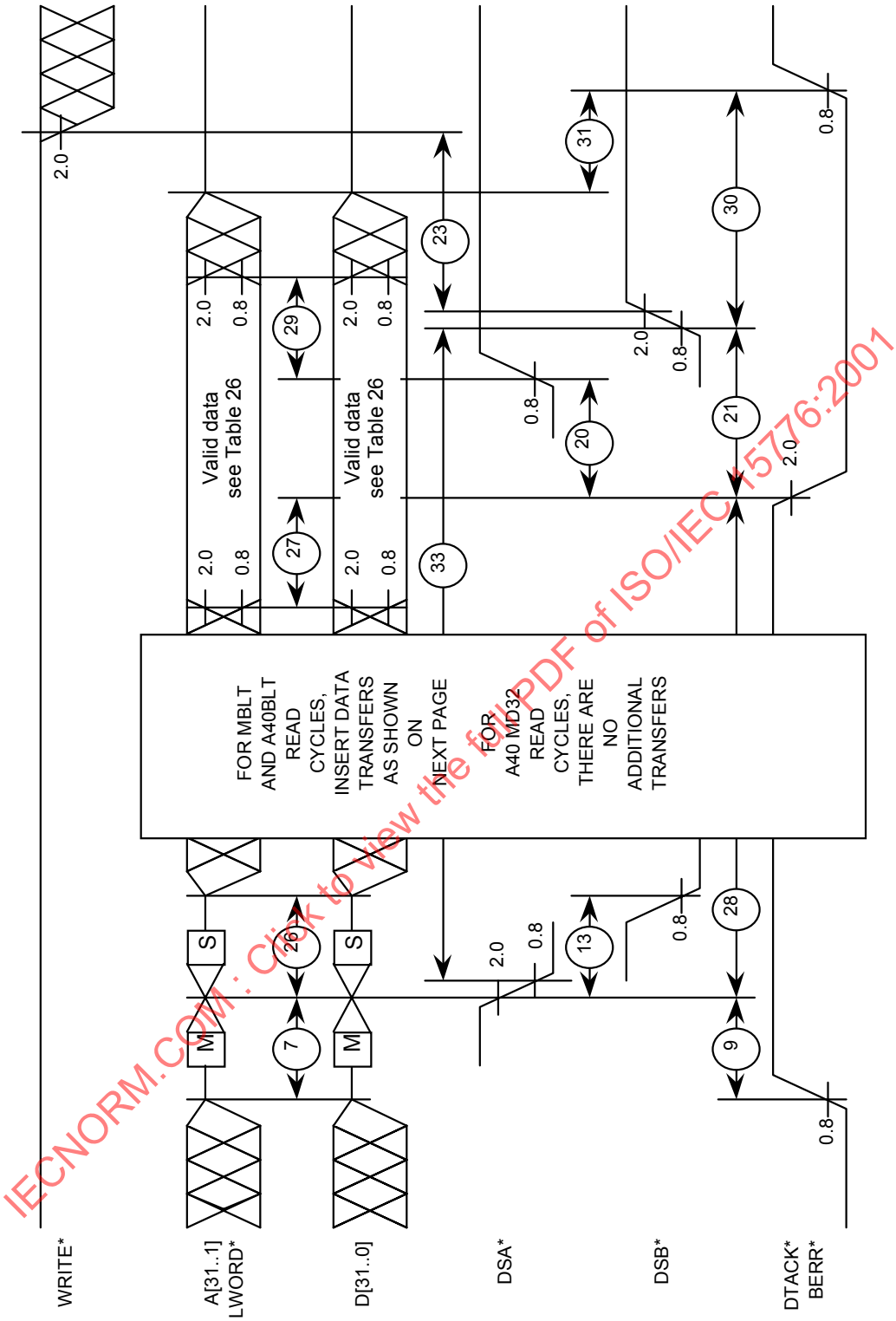
Byte(0-1) Read, Byte(2-3) Read

Byte(0-3) Read, Byte(1-2) Read

Double-Byte Block Read

Quad-Byte Block Read

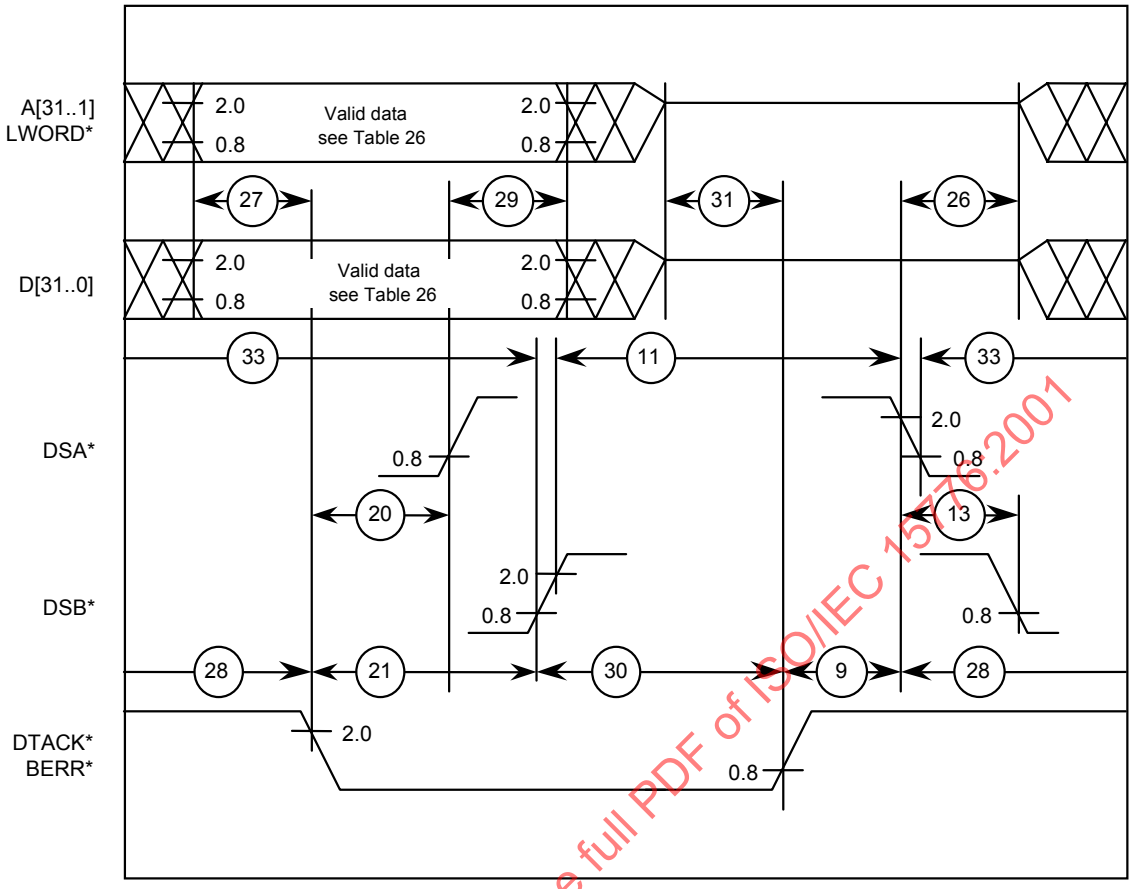
Figure 23 – Master, slave and location monitor data transfer timing (*continued*)



NOTE 1 The rising and falling edge of DSA* and DSB* are shown disconnected to indicate that the signal that is the first to fall is not necessarily the one that is the first to rise.

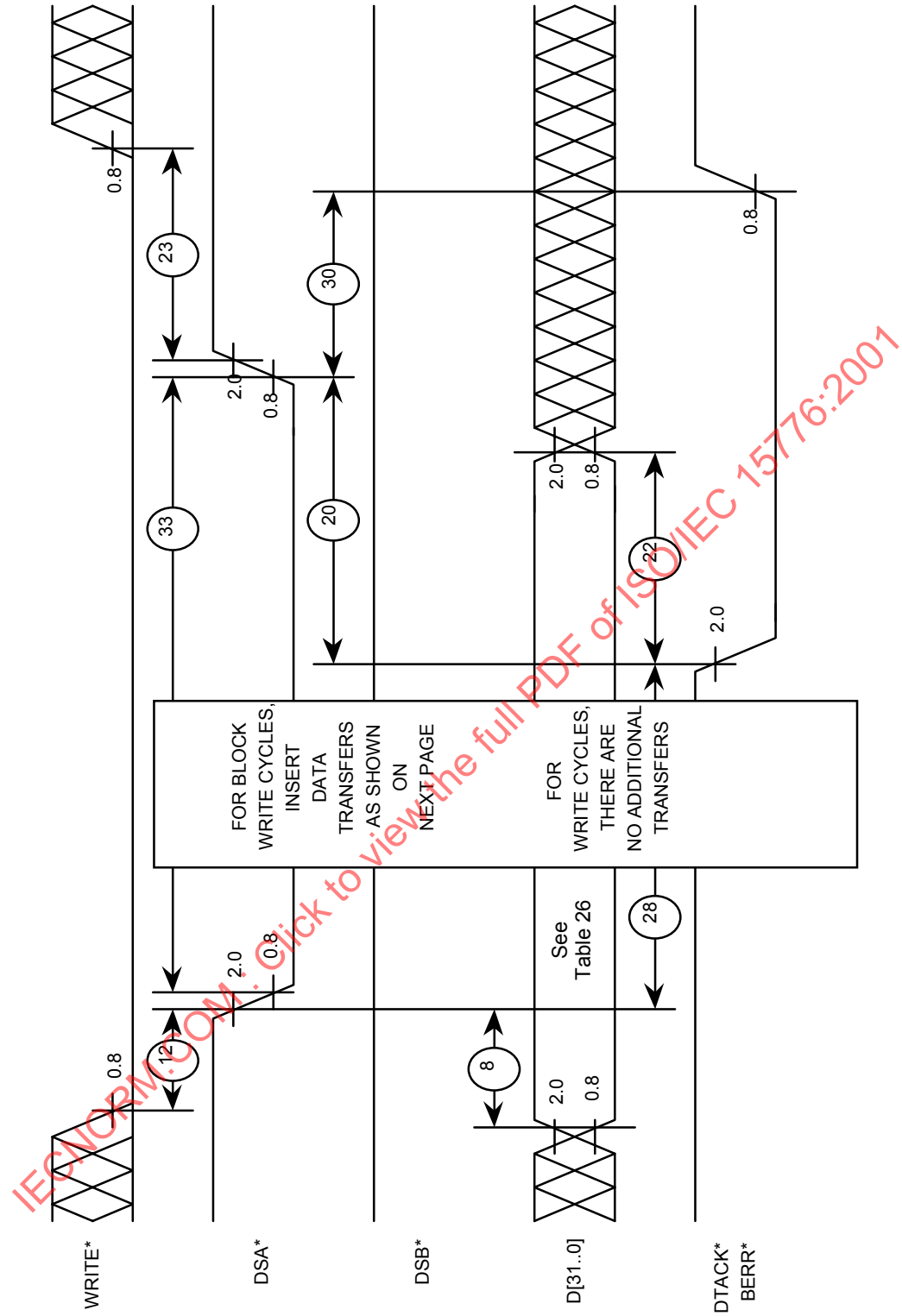
NOTE 2 See Figure 21 for address broadcast timing.

Figure 24 – Master, slave and location monitor data transfer timing A40 multiplexed quad byte read, A40BLT multiplexed quad byte block read, MBLT eight byte block read



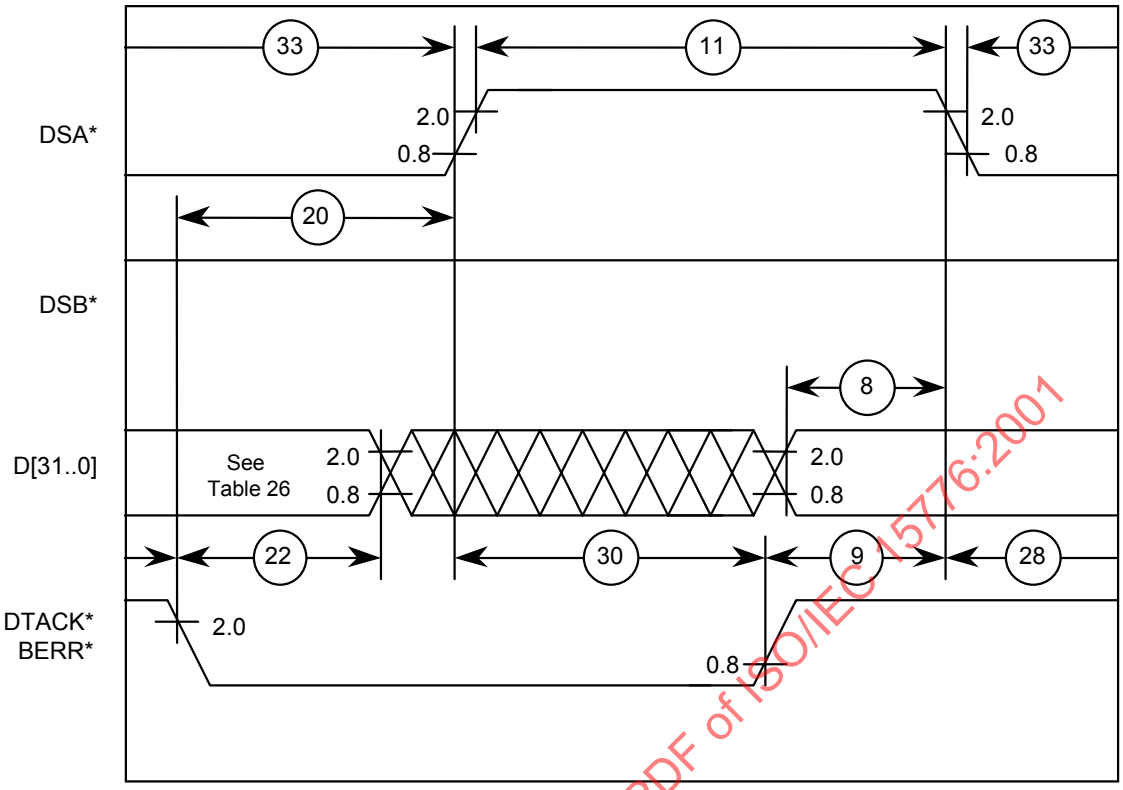
NOTE The rising and falling edges of DSA* and DSB* are not connected to indicate that the signal that is the first to rise is not necessarily the one that is the first to fall.

Figure 24 – Master, slave and location monitor data transfer timing A40 multiplexed quad byte read, A40BLT multiplexed quad byte block read, MBLT eight byte block read (*continued*)



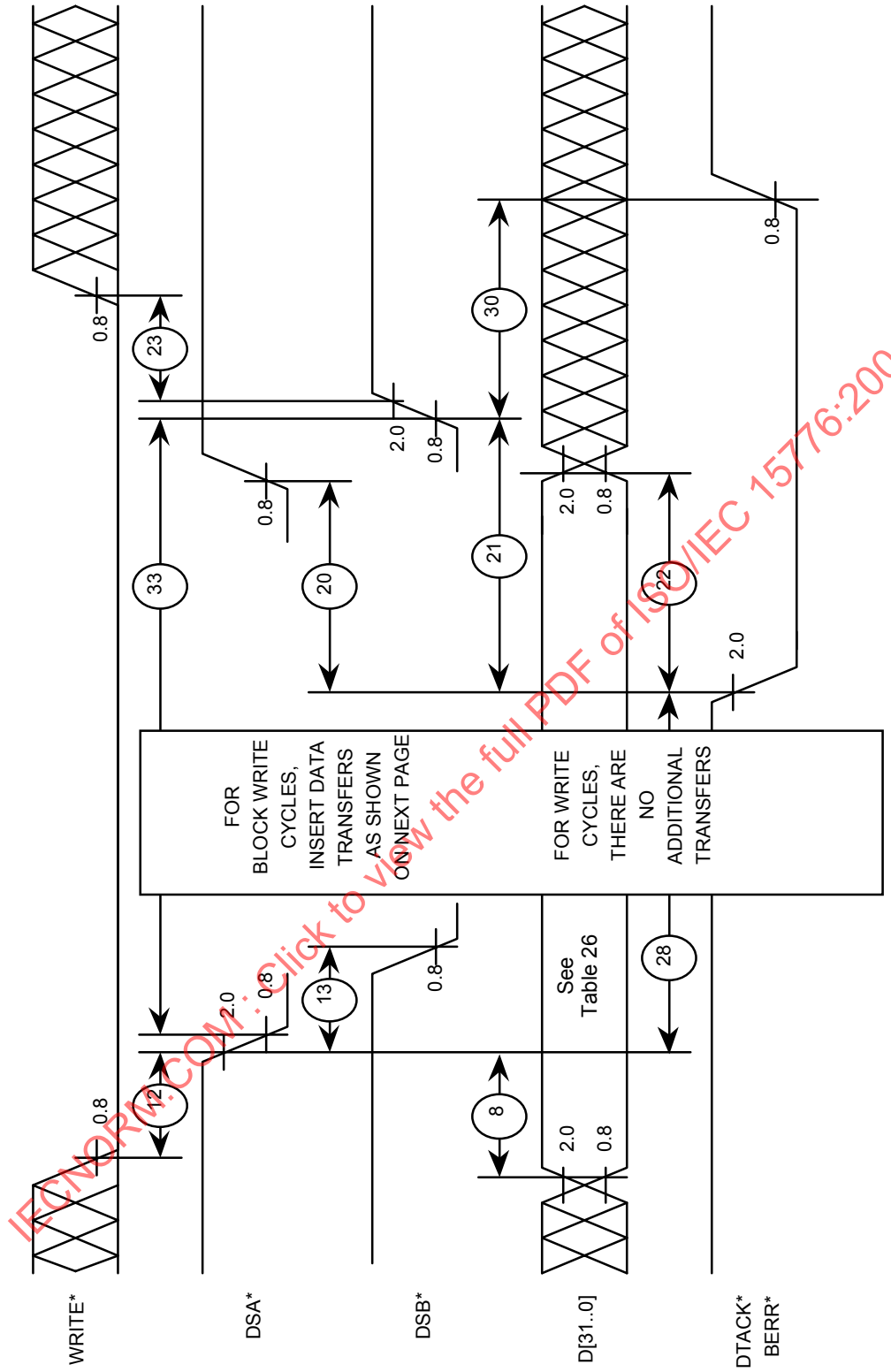
Byte(0) Write, Byte(1) Write
Byte(2) Write, Byte(3) Write
Byte(0-2) Write, Byte(1-3) Write
Single-Byte Block Write

Figure 25 – Master, slave and location monitor data transfer timing



Byte(0) Write, Byte(1) Write
Byte(2) Write, Byte(3) Write
Byte(0-2) Write, Byte(1-3) Write
Single-Byte Block Write

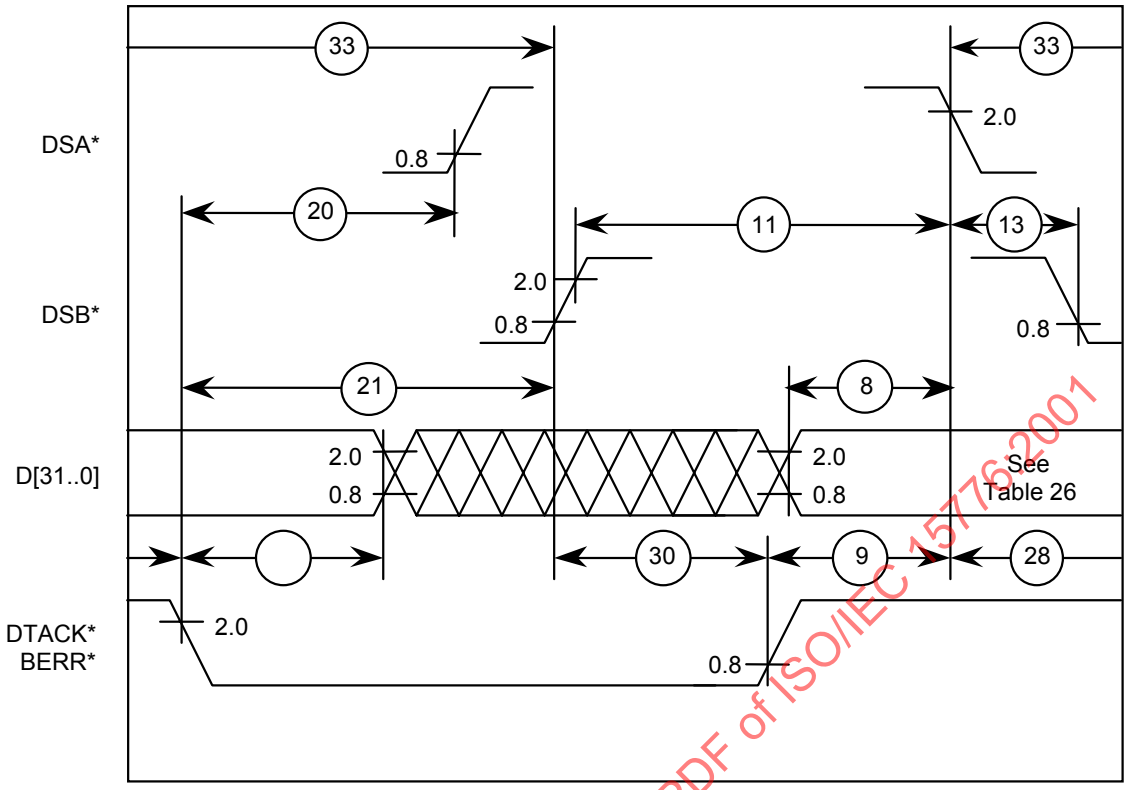
Figure 25 – Master, slave and location monitor data transfer timing (*continued*)



NOTE The falling and rising edges of DSA* and DSB* are not connected to indicate that the signal that is the first to fall is not necessarily the one that is the first to rise.

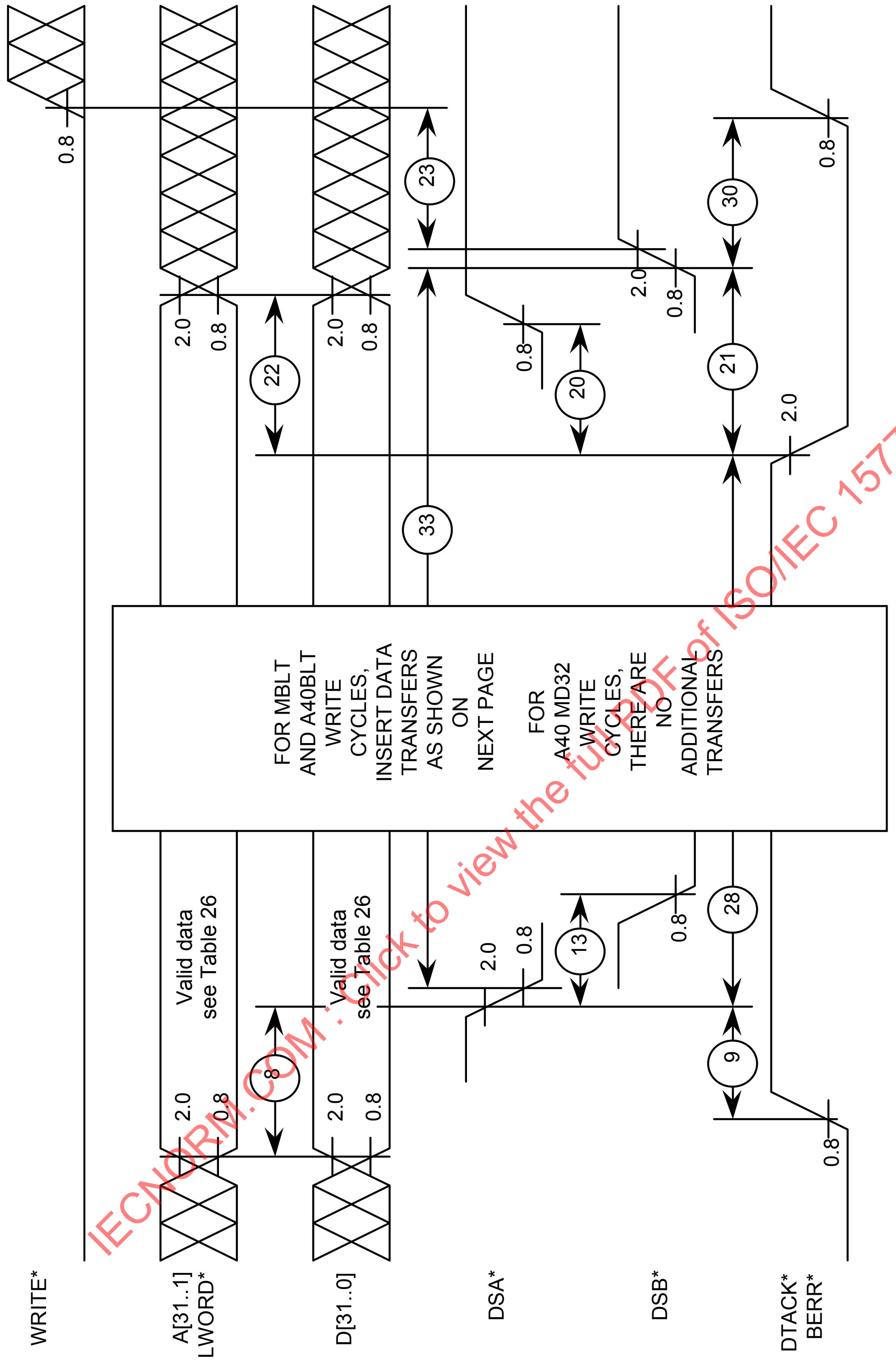
Byte(0-1) Write, Byte(2-3) Write
Byte(0-3) Write, Byte(1-2) Write
DOUBLE-Byte Block Write
Quad-Byte Block Write

Figure 26 – Master, slave and location monitor data transfer timing



Byte(0-1) Write, Byte(2-3) Write
Byte(0-3) Write, Byte(1-2) Write
DOUBLE-Byte Block Write
Quad-Byte Block Write

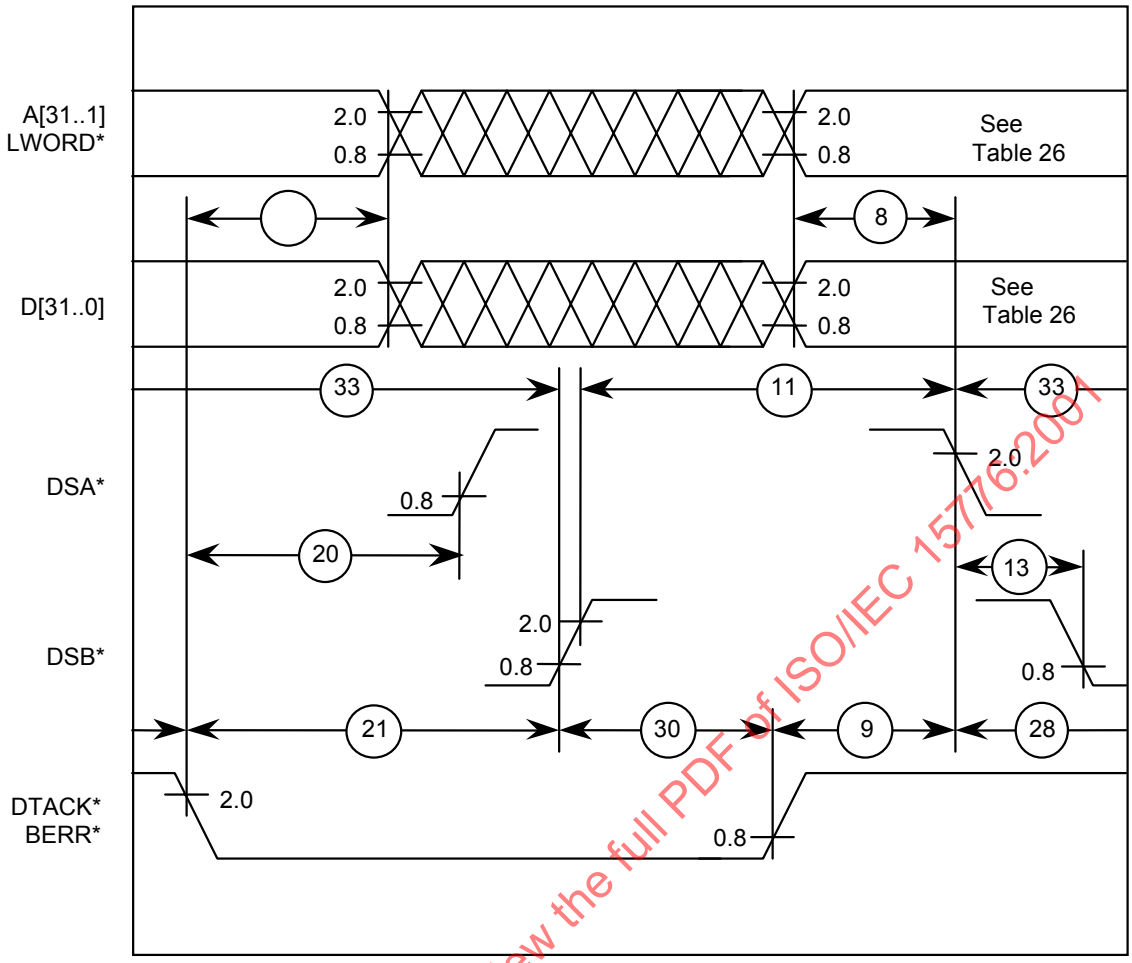
Figure 26 – Master, slave and location monitor data transfer timing (*continued*)



NOTE 1 The rising and falling edge of DSA* and DSB* are shown disconnected to indicate that the signal that is the first to fall is not necessarily the one that is the first to rise.

NOTE 2 See Figure 21 for address broadcast timing.

Figure 27 – Master, slave and location monitor data transfer timing A40 multiplexed quad byte write, A40BLT multiplexed quad byte block write, MBLT eight byte block write



NOTE The rising and falling edges of DSA* and DSB* are not connected to indicate that the signal that is the first to rise is not necessarily the one that is the first to fall.

Figure 27 – Master, slave and location monitor data transfer timing A40 multiplexed quad byte write, A40BLT multiplexed quad byte block write, MBLT eight byte block write (*continued*)

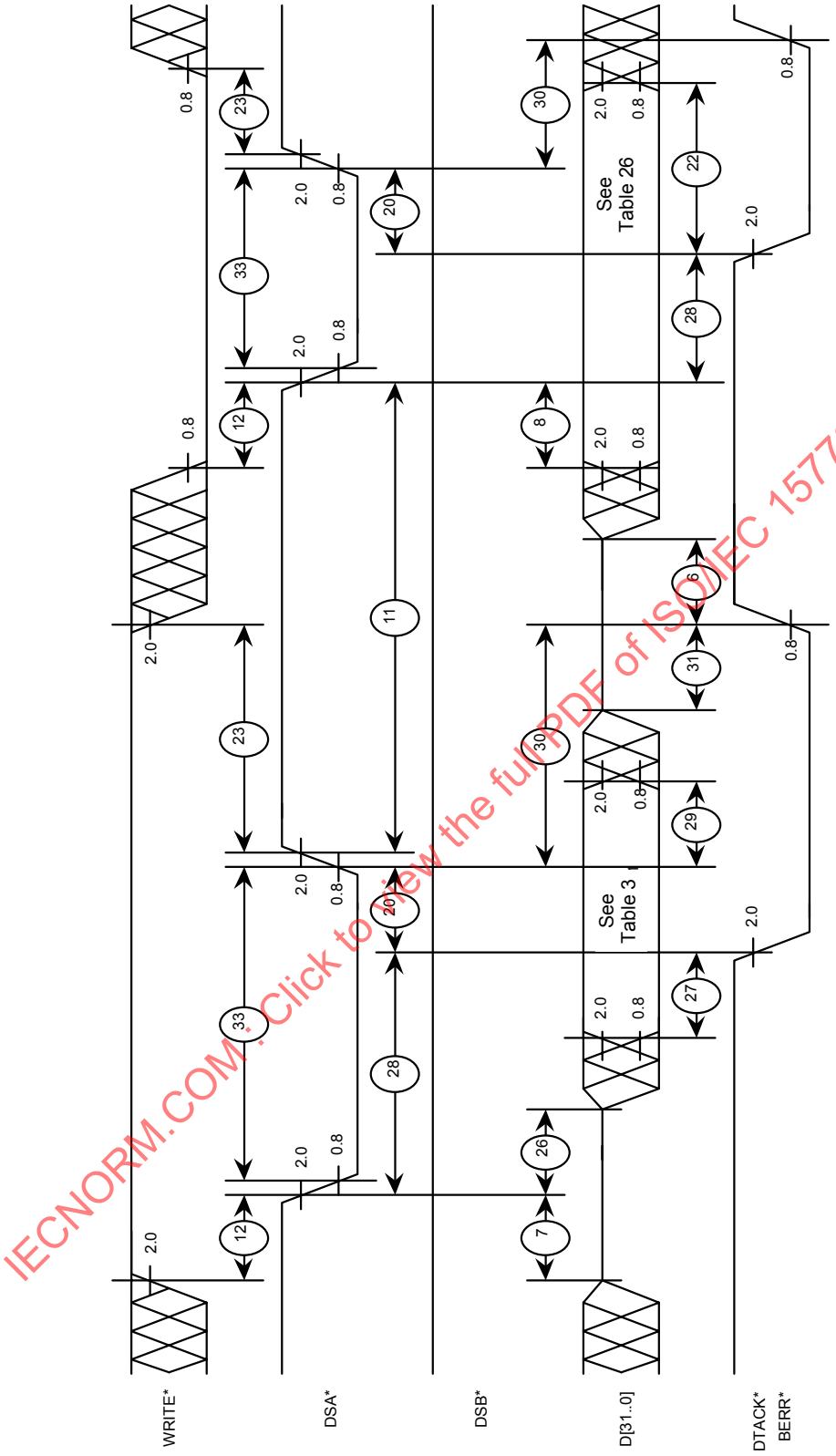
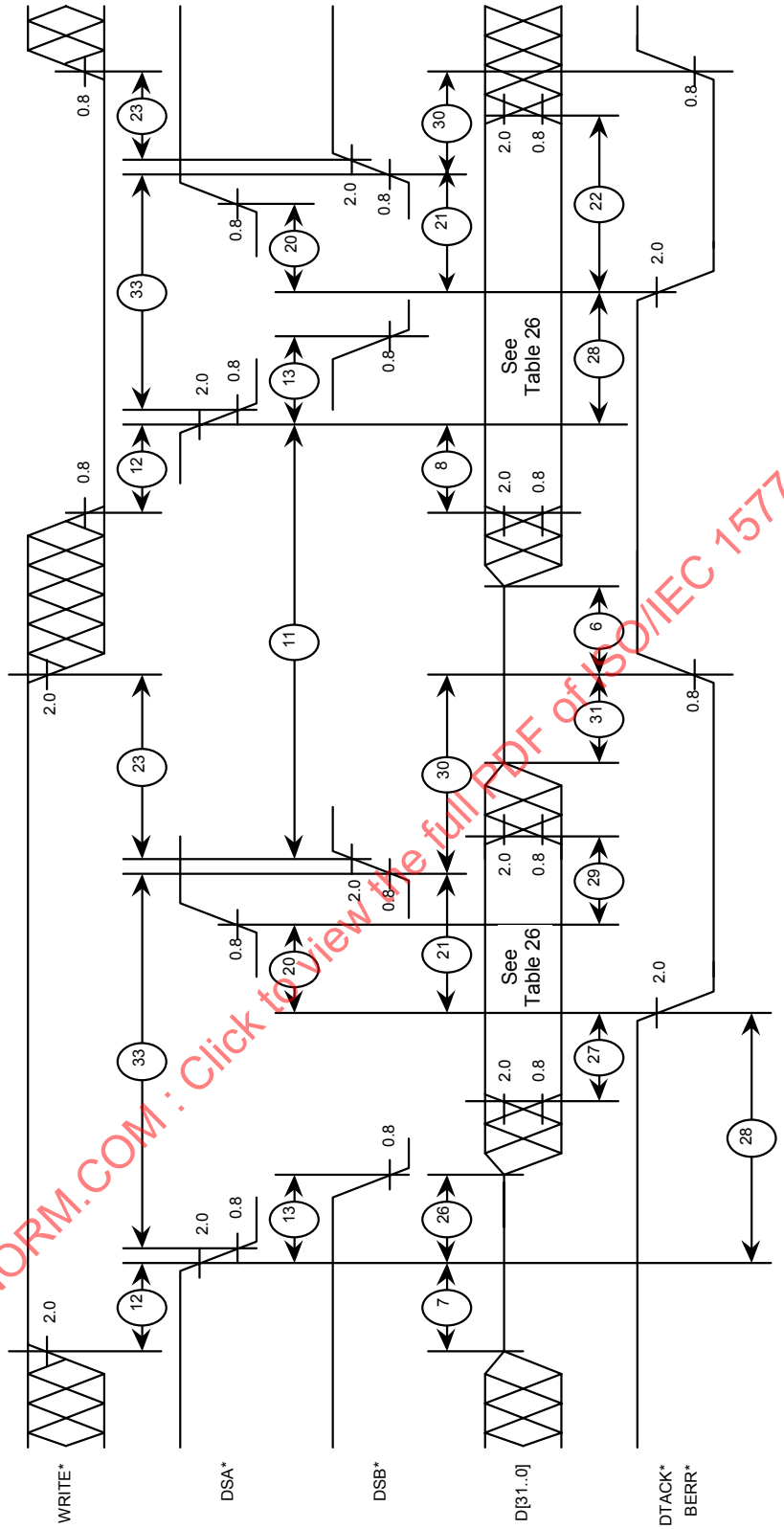


Figure 28 – Master, slave and location monitor data transfer timing single-byte RMW cycles



NOTE The rising and falling edges of DSA* and DSB* are not connected to indicate that the signal that is the first to rise is not necessarily the one that is the first to fall.

Figure 29 – Master, slave and location monitor data transfer timing double-byte RMW cycles, quad-byte RMW cycles

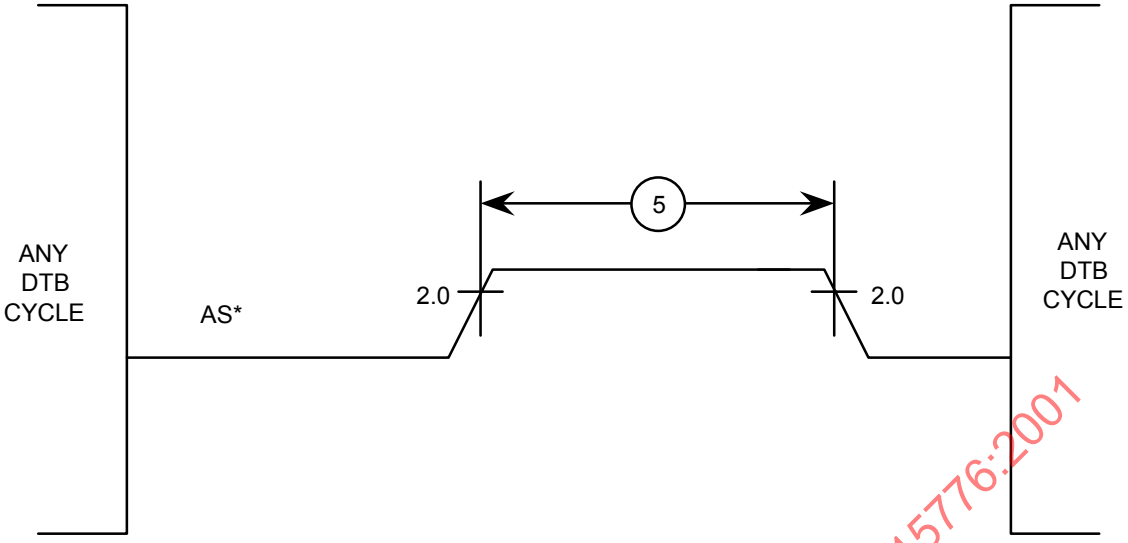
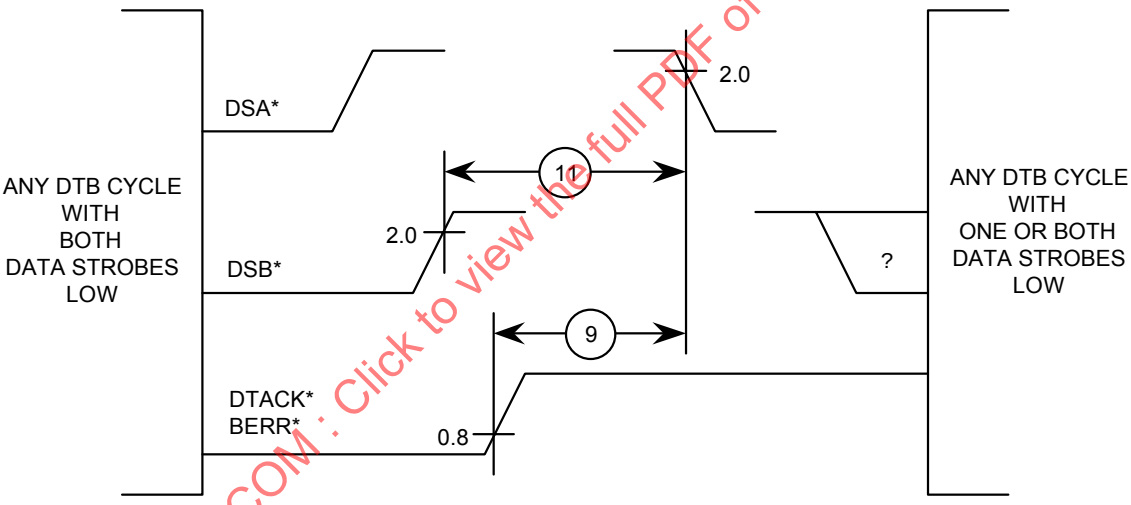
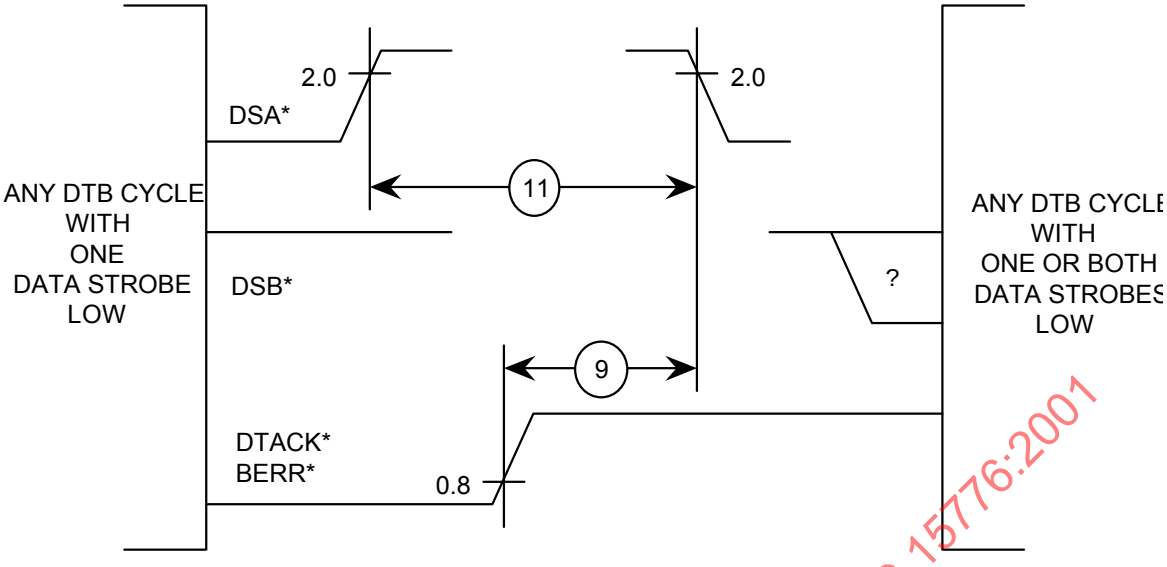


Figure 30 – Address strobe inter-cycle timing



An example of a cycle where both data strobes go low followed by a cycle where one or both data strobes go low.

Figure 31 – Data strobe inter-cycle timing



An example of a cycle where one data strobe goes low, followed by a cycle where one or both data strobes go low.

Figure 32 – Data strobe inter-cycle timing

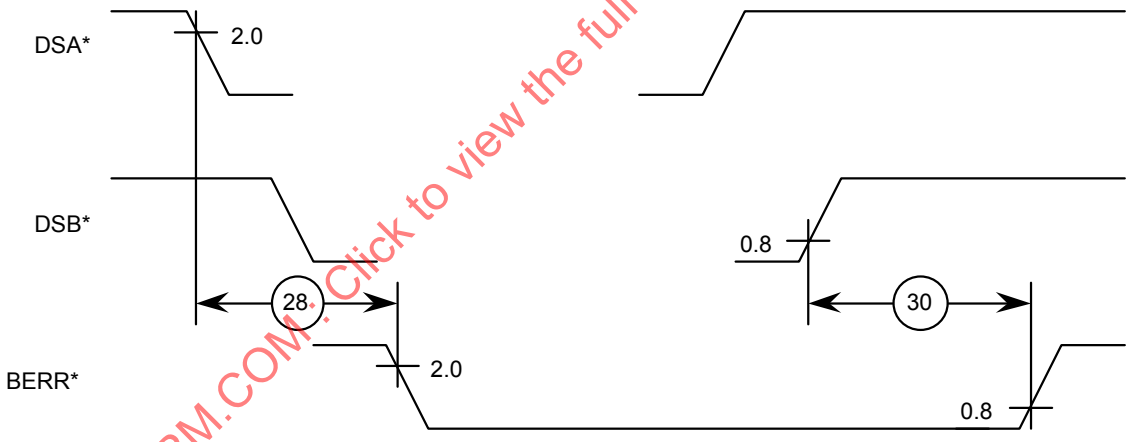


Figure 33 – Master, slave and bus timer data transfer timing timed-out cycle

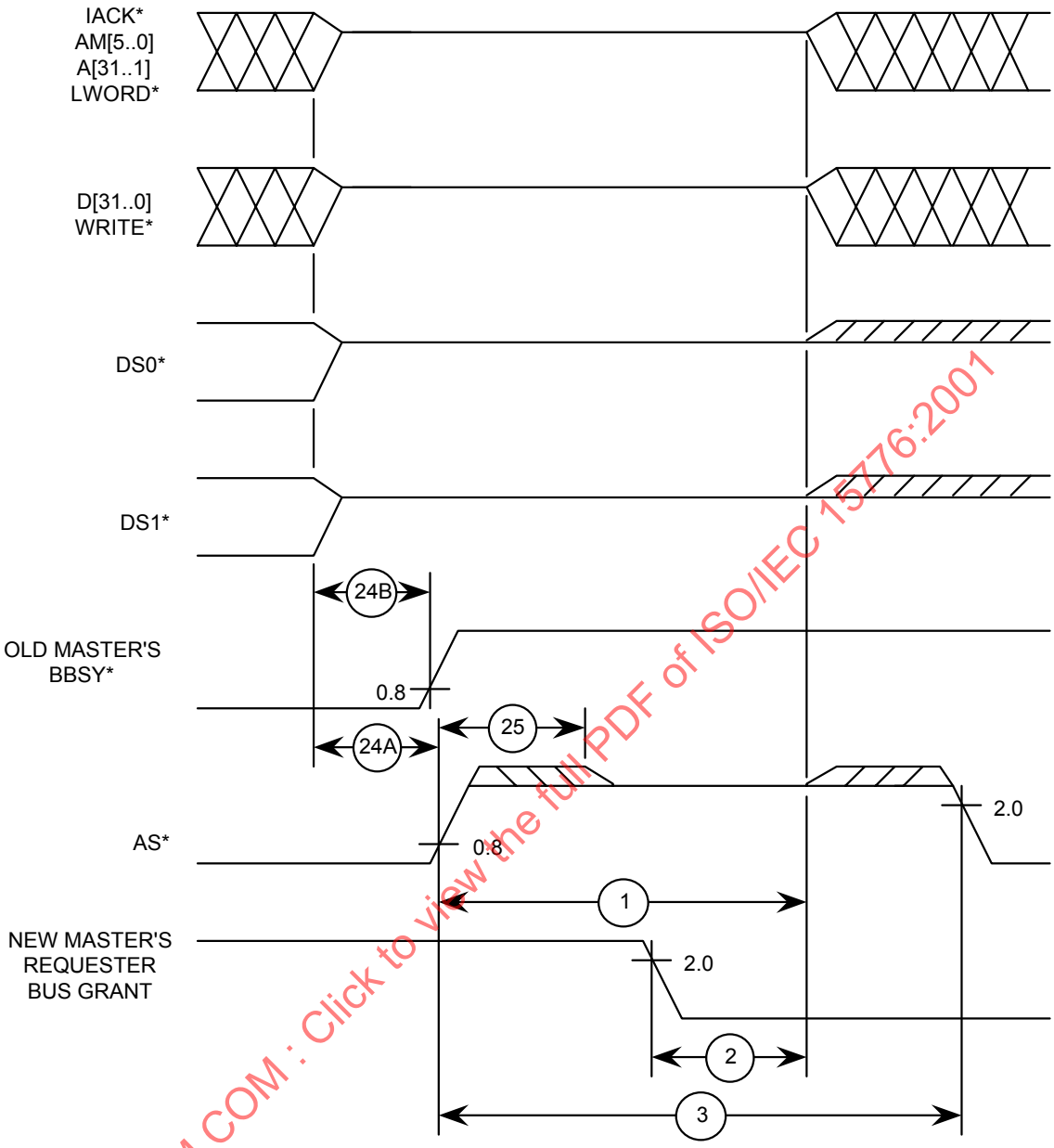
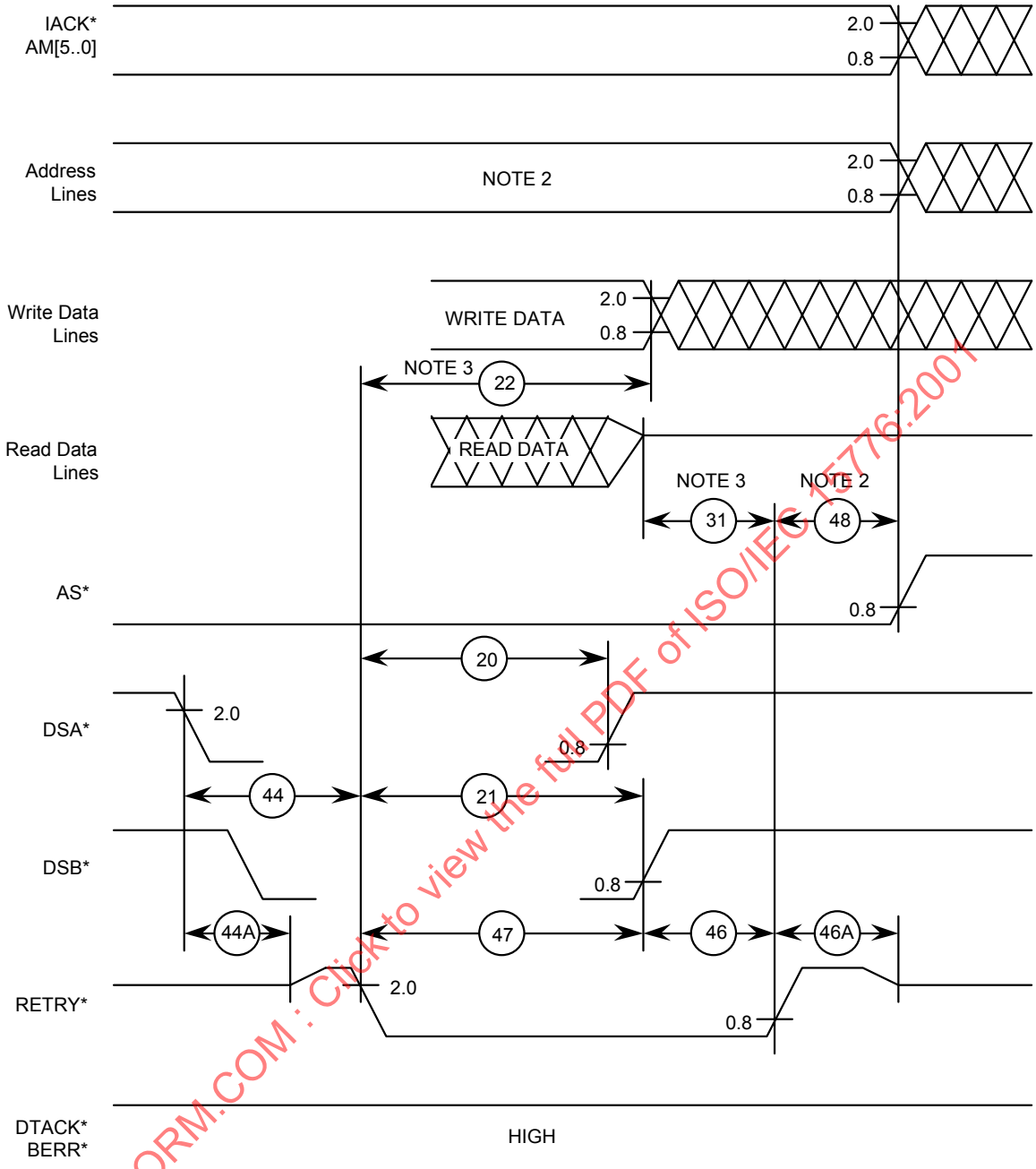


Figure 34 – Master DTB control transfer timing

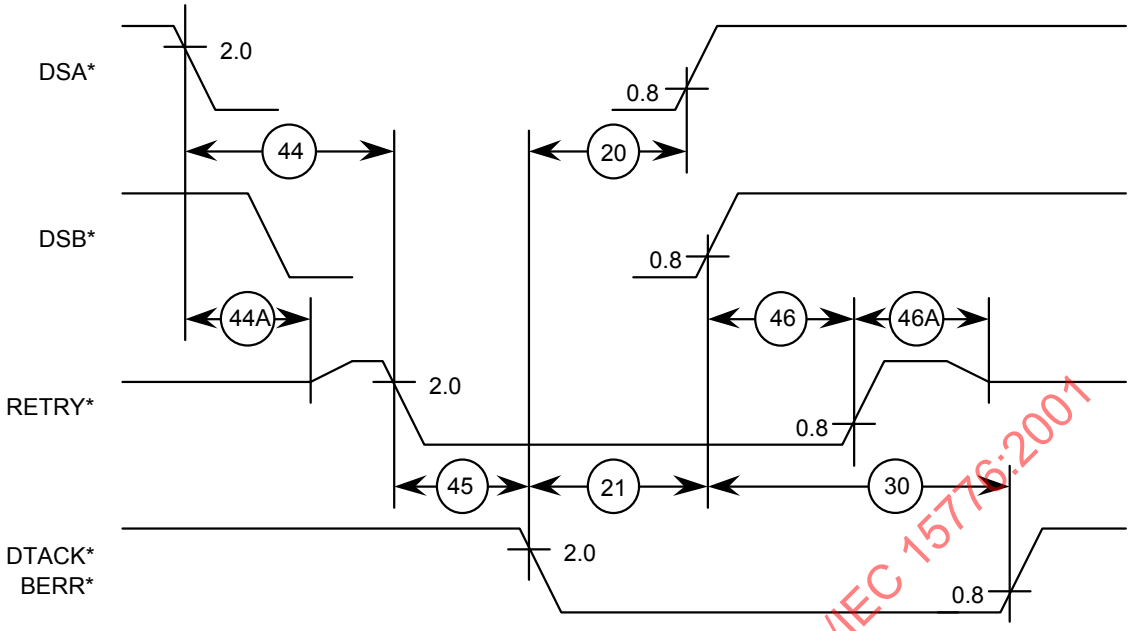


NOTE 1 The rising and falling edges of DSA* and DSB* are not connected to indicate that the signal that is the first to rise is not necessarily the one that is the first to fall.

NOTE 2 Timing parameter 48 does not apply to the address lines during the data phase of multiplexed-address or multiplexed-data cycles. Nor does it imply any timing relationships between AS* and address information.

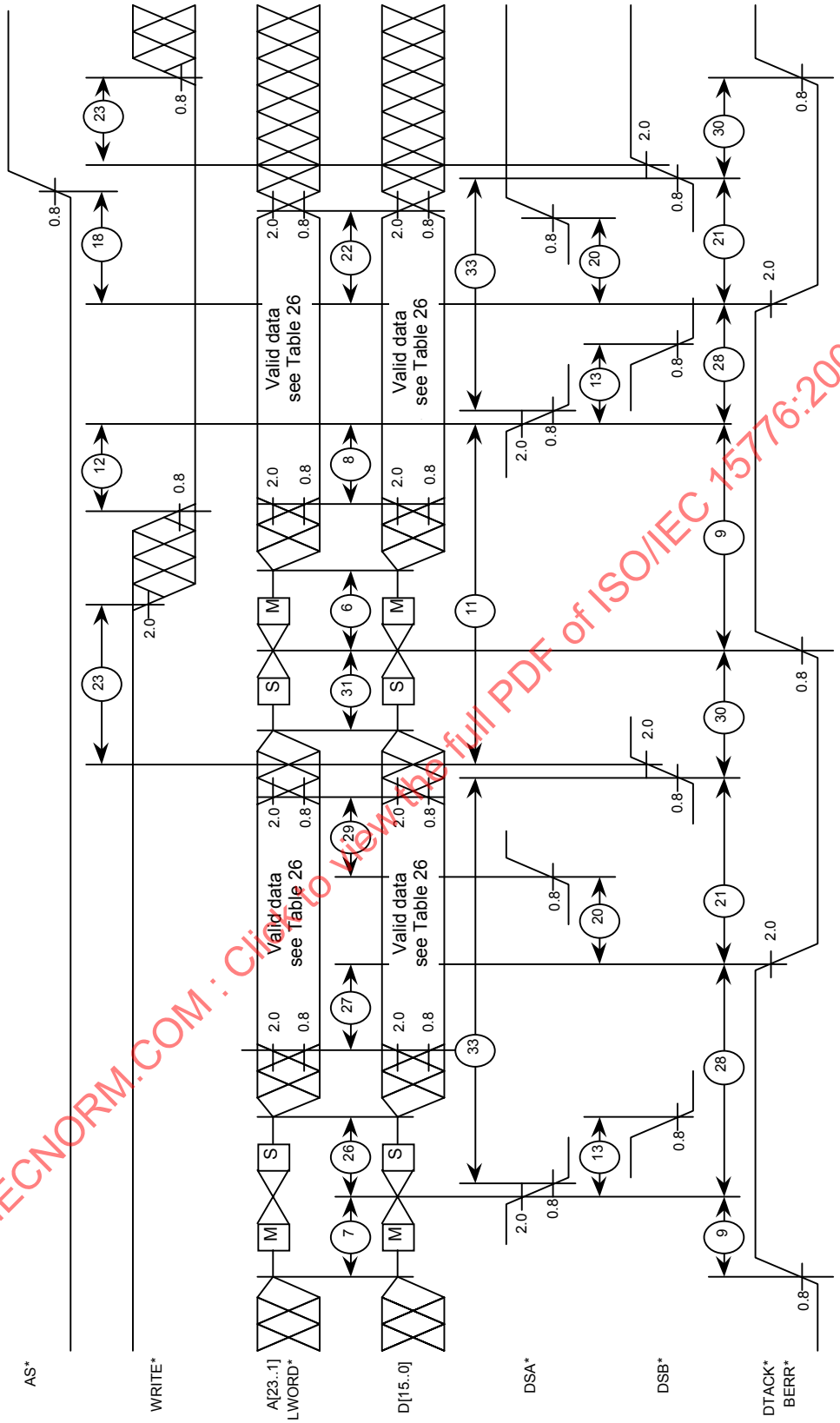
NOTE 3 Apply timing parameter 22 for write cycles and parameter 31 for read cycles.

Figure 35 – Master and slave data transfer timing master responding to RETRY* line



NOTE The rising and falling edges of DSA* and DSB* are not connected to indicate that the first rising signal is not necessarily the first to fall.

Figure 36 – Master and slave data transfer timing master ignoring RETRY* line



NOTE 1 The rising and falling edges of DSA* and DSB* are shown disconnected to indicate that the signal that is the first to fall is not necessarily the one that is the first to rise.

NOTE 2 See figure 2-27 for address broadcast timing.

Figure 37 – A40, MD32 Read-Modify-Write

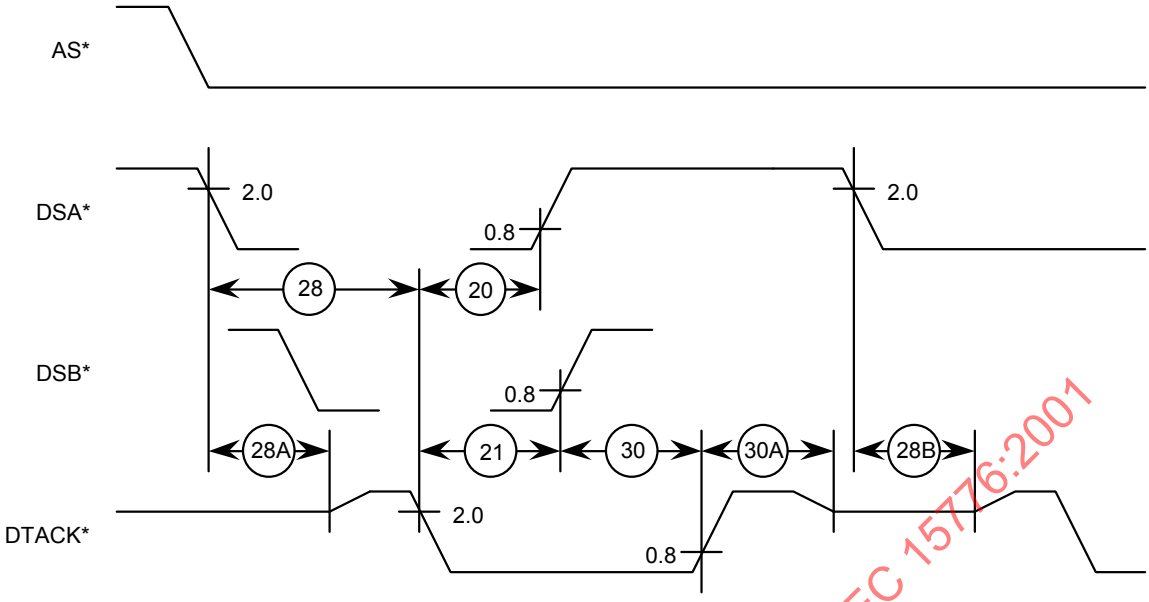


Figure 38 – Rescinding DTACK timing

3 Data transfer bus arbitration

3.1 Bus arbitration philosophy

As microprocessor costs decrease, it is becoming more cost effective to design systems with multiple processors sharing global resources.

The most fundamental of these global resources is the Data Transfer Bus through which all other global resources are accessed. Therefore, any system that supports multiprocessing needs to provide an efficient allocation method for the Data Transfer Bus. Because speed of allocation is vital, a hardware allocation scheme is the only practical alternative. The VMEbus meets this need with its Arbitration subsystem. (See Figure 39.)

The VMEbus arbitration subsystem

a) prevents simultaneous use of the bus by two Masters,

b) schedules requests from multiple Masters for optimum bus use.

3.1.1 Types of arbitration

When several boards request use of the DTB simultaneously, the arbitration subsystem detects these requests and grants the bus to one board at a time. The decision of which board is granted the bus first depends upon what scheduling algorithm is used. Many algorithms are possible. This specification describes four arbitration algorithms: prioritized, round-robin, single level and fair arbitration.

Prioritized arbitration assigns the bus according to a fixed priority scheme where each of four bus request lines has a priority from highest (BR3*) to lowest (BR0*).

Round-robin arbitration assigns the bus on a rotating priority basis. When the bus is granted to the Requester on bus request line BR(n)*, then the highest priority for the next arbitration is assigned to bus request line BR(n-1)*.

Single level arbitration only accepts requests on BR3*, and relies on BR3*'s bus grant daisy-chain to arbitrate the requests.

Fair arbitration ensures that all requesters on the same request level get equal access to the DTB. In the fair arbitration algorithm a requester will not issue a request for the DTB if a request has been issued by another requester on its own level.

PERMISSION 3.1

Scheduling algorithms other than priority, round-robin, or single level MAY be used. For example, an Arbiter's algorithm might give highest priority to BR3*, but grant the bus to BR0* through BR2* on a round-robin basis.

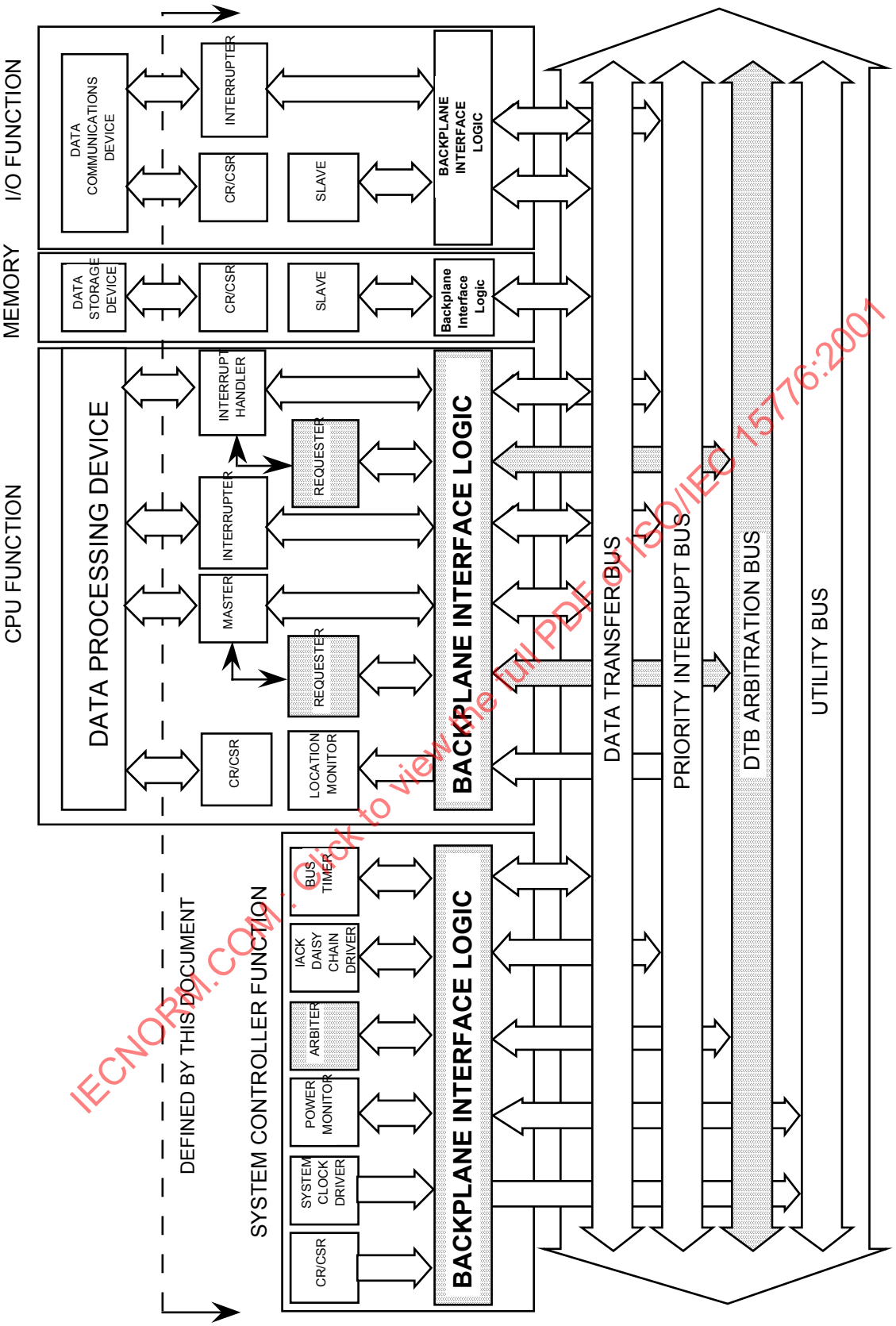


Figure 39 – Arbitration functional block diagram

3.2 Arbitration bus lines

The Arbitration Bus consists of six bused VMEbus lines and four daisy-chained lines. These daisy-chained lines require special signal names. The signals entering each board are called “Bus Grant IN” lines (BG[3..0]IN*), while the signals leaving each board are called “Bus Grant OUT” lines (BG[3..0]OUT*). The lines which leave slot n as BG[3..0]OUT* enter slot n+1 as BG[3..0]IN*. This is illustrated in Figure 40.

OBSERVATION 3.1

In all descriptions in this chapter, the terms BRx*, BG[3..0]IN*, and BG[3..0]OUT* are used to describe the bus request and bus grant lines, where x takes on any value from zero to three.

In the VMEbus arbitration system, a Requester module drives the following lines:

1 bus request line	(one of BR0* through BR3*)
1 bus grant out line	(one of BG0OUT* through BG3OUT*)
1 bus busy line	(BBSY*)

RULE 3.1

IF a VMEbus board does not generate bus requests on some bus request levels,

THEN it **MUST** propagate the daisy-chain signals for those levels from its BG[3..0]IN* lines to its respective BG[3..0]OUT* lines.

PERMISSION 3.2

The propagation for the unused lines of the bus grant daisy-chain MAY be done using jumpers or active logic. The latter approach allows selection of the request level under software control, while the former results in faster propagation through the daisy-chain.

Three types of Arbiters are described in this standard. They are:

PRIoritized	(PRI)
Round-Robin-Select	(RRS)
SinGLe level	(SGL)

The operation of these three types of Arbiters is described in 3.3.

A PRI Arbiter drives the following:

1 bus clear line	(BCLR*)
4 bus grant lines	(Slot 1 BG0IN* through BG3IN*) if the Arbiter’s board also has a Requester.
	(Slot 1 BG0OUT* through BG3OUT*) if the Arbiter’s board does not have a Requester.

An RRS Arbiter drives the four Slot 1 BG[3..0]IN* or BG[3..0]OUT* lines and, optionally, the BCLR* line.

A SGL Arbiter drives only BG3IN* or BG3OUT* at Slot 1, ad, optionally the BCLR* line.

Two additional lines are connected (although indirectly) with the arbitration system during power-up and power-down sequencing: SYSRESET* and ACFAIL*. While their impact on the arbitration system is included in this clause, these lines will be discussed further in clause 5.

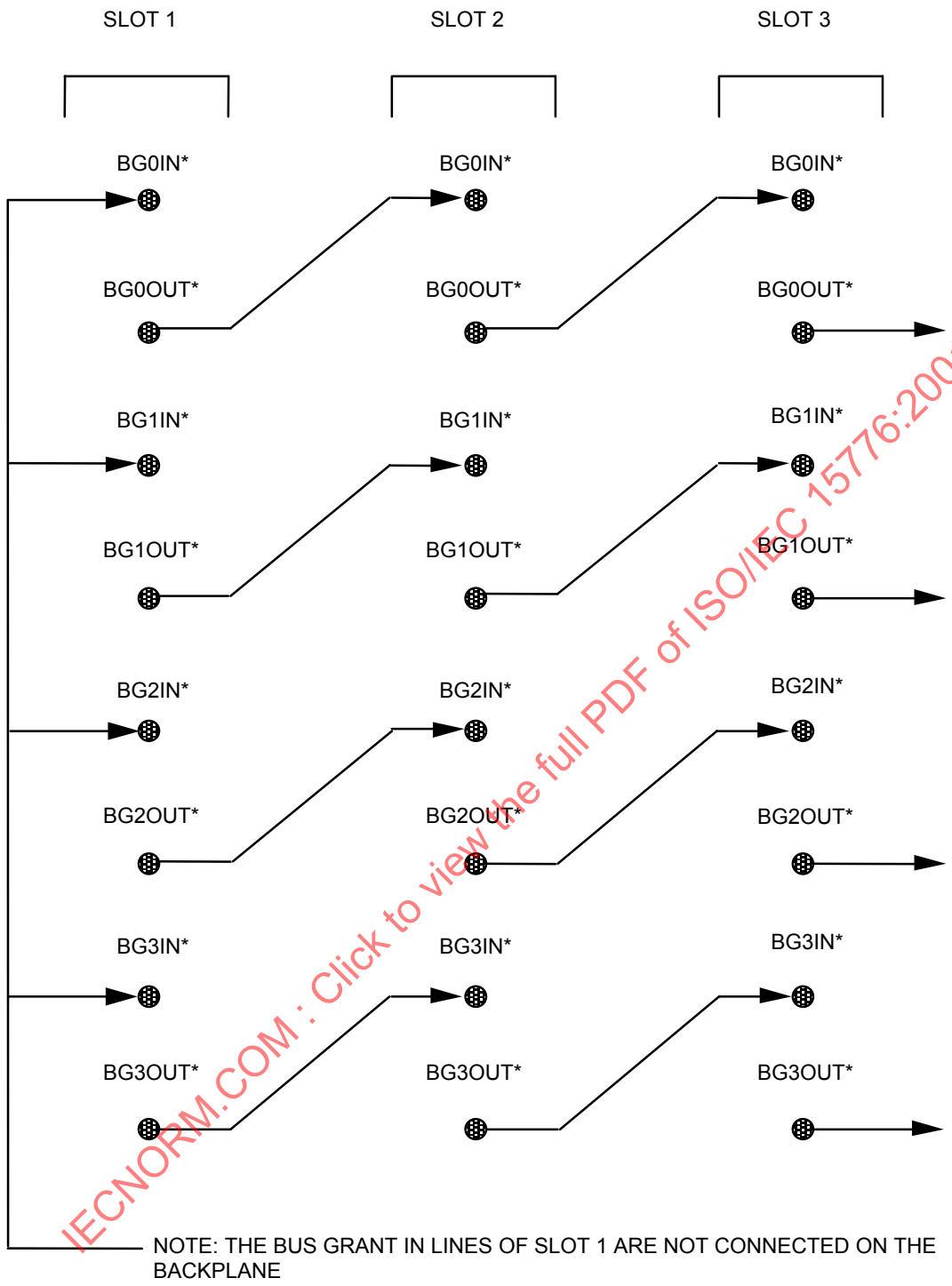


Figure 40 – Illustration of the daisy chain bus grant lines

3.2.1 Bus request and bus grant lines

The bus request lines are used by each Requester to request use of the DTB. The bus grant lines allow the Arbiter to award use of the bus. It does this by driving a bus grant daisy-chain line low. This low level propagates down the daisy-chain, typically passing through several boards in the process. If a board never uses a particular request/grant level, the signal is passed through that board. When the board uses a request/grant level x, the corresponding signal BG[3..0]IN* is gated on board. If its on-board Requester is currently requesting the DTB on that level, it does not pass the low level on to its BG[3..0]OUT*. Otherwise, it passes on the low level to the next slot in the chain.

RULE 3.2

IF a VMEbus backplane slot is not occupied by a board, and if there are boards farther down the daisy-chain,

THEN jumpers **MUST** be installed at the empty slot to pass through all daisy-chain signals.

OBSERVATION 3.2a

The backplane mechanical specification in clause 7 describes a provision for the installation of jumpers at each slot. It is possible to provide automatic mechanical or electronic jumpering on VMEbus backplanes.

RULE 3.3

The Arbiter **MUST** always be located in Slot 1.

3.2.2 Bus busy line (BBSY*)

Once a Requester has been granted control of the Data Transfer Bus via the bus grant daisy-chain, it drives BBSY* low. It then has control of the DTB until it releases BBSY*, allowing the Arbiter to grant the DTB to some other Requester.

3.2.3 Bus clear line (BCLR*)

The PRI Arbiter drives BCLR* low to inform the Master, currently in control of the DTB, when a higher priority request is pending. The current Master does not have to relinquish the bus within any prescribed time limit. It can continue transferring data until it reaches an appropriate stopping point, and then allow its on-board Requester to release BBSY*.

PERMISSION 3.3

Although RRS Arbiters are not required to drive the BCLR* line they MAY do so.

SUGGESTION 3.1

IF an RRS Arbiter drives the BCLR* line low.

THEN design it to do so whenever there is a request pending on any of the non-granted bus request lines

3.3 Functional modules

The arbitration subsystem is composed of several modules:

a) One Arbiter;

b) One or more Requesters.

Figures 41 and 42 provide block diagrams for the two types of Arbitration Bus modules.

RULE 3.4

Output signal lines shown with solid lines in Figure 41 and 42 **MUST** be driven by the module, unless it would always drive them high.

RULE 3.5

Input signal lines shown with solid lines in Figures 41 and 42 **MUST** be monitored and responded to in the appropriate fashion.

OBSERVATION 3.3

RULEs and PERMISSIONs for driving and monitoring the signal lines shown with dotted lines in Figures 41 and 42 are given in Tables 33 and 34.

OBSERVATION 3.4

IF a bused output signal line is not driven

THEN terminators on the backplane ensure that it is high.

OBSERVATION 3.5

Although SYSRESET* and ACFAIL* are not specified as part of the Arbitration Bus, they are important here because Masters, which are paired with Requesters, respond to these signal lines. (SYSRESET* and ACFAIL* are driven by the Power Monitor module which is discussed in clause 5.)

3.3.1 Arbiter

The Arbiter is a functional module that decides which Requester should be granted control of the DTB when several Requesters request it simultaneously. There are many possible algorithms that could be used to make this decision. Three types of Arbiters are described in this specification: a prioritized (PRI) Arbiter, a round-robin (RRS) Arbiter, and a single level (SGL) Arbiter.

An Arbiter responds to incoming bus requests and grants the DTB to the appropriate Requester with one of the bus grant lines.

When the Arbiter detects BBSY* high for at least 40 ns, and after it detects one or more bus requests, it issues a bus grant, corresponding to the highest priority bus request.

When the Requester receives the bus grant, it drives BBSY* low and signals to its on-board Master or Interrupt Handler that it has been granted the DTB. After its on-board Master or Interrupt Handler finishes using the DTB, the Requester releases BBSY*. The resulting rising edge of BBSY* signals the Arbiter to issue another bus grant, based upon the levels of the bus request lines at that time.

In addition to the arbitration provided by the Arbiter, a secondary level of arbitration is provided by the bus grant daisy-chains. Because of these daisy-chains, Requesters sharing a common request line are prioritized by slot position. The Requester closest to Slot 1 has the highest priority.

SGL Arbiters respond only to bus requests on BR3* and depend on the BG3IN*/BG3OUT* daisy-chain to do the priority arbitration.

The PRI Arbiter prioritizes the four bus request lines, from BR0* (the lowest) to BR3* (the highest), and responds with BG0IN* through BG3IN*, as appropriate. A PRI Arbiter also informs any Master, currently in control of the bus, when a higher level request is pending by driving BCLR* low.

To visualize an RRS Arbiter, consider a mechanical switch being driven by a stepping motor. Each position on the switch connects a bus request line to its corresponding bus grant line. When the bus is busy, the switch is stopped on the current level. Upon release of the bus, the switch steps one position lower (i.e., from BR(n)* to BR(n-1)*) and tests for a request. It continues this scanning operation until a request is found, sending a bus grant over the appropriate line.

PERMISSION 3.4

An Arbiter MAY be designed with a built-in time-out feature that causes it to withdraw a bus grant if BBSY* is not driven low by a Requester within a prescribed time.

OBSERVATION 3.6

The time used by the Arbiter allowed by PERMISSION 3.4 needs to be longer than the longest possible bus grant daisy-chain propagation delay time, plus the time the slowest Requester takes to generate BBSY*.

RULE 3.6

Except for a time-out situation where no Requester responds, once the Arbiter grants the bus to a Requester it **MUST NOT** generate a new bus grant before that Requester generates a rising edge on the BBSY* line. (The Requester generates a rising edge by driving BBSY* low and then releasing it.)

OBSERVATION 3.7

IF an Arbiter uses a “snapshot” of the request lines taken prior to the rising edge of BBSY*.

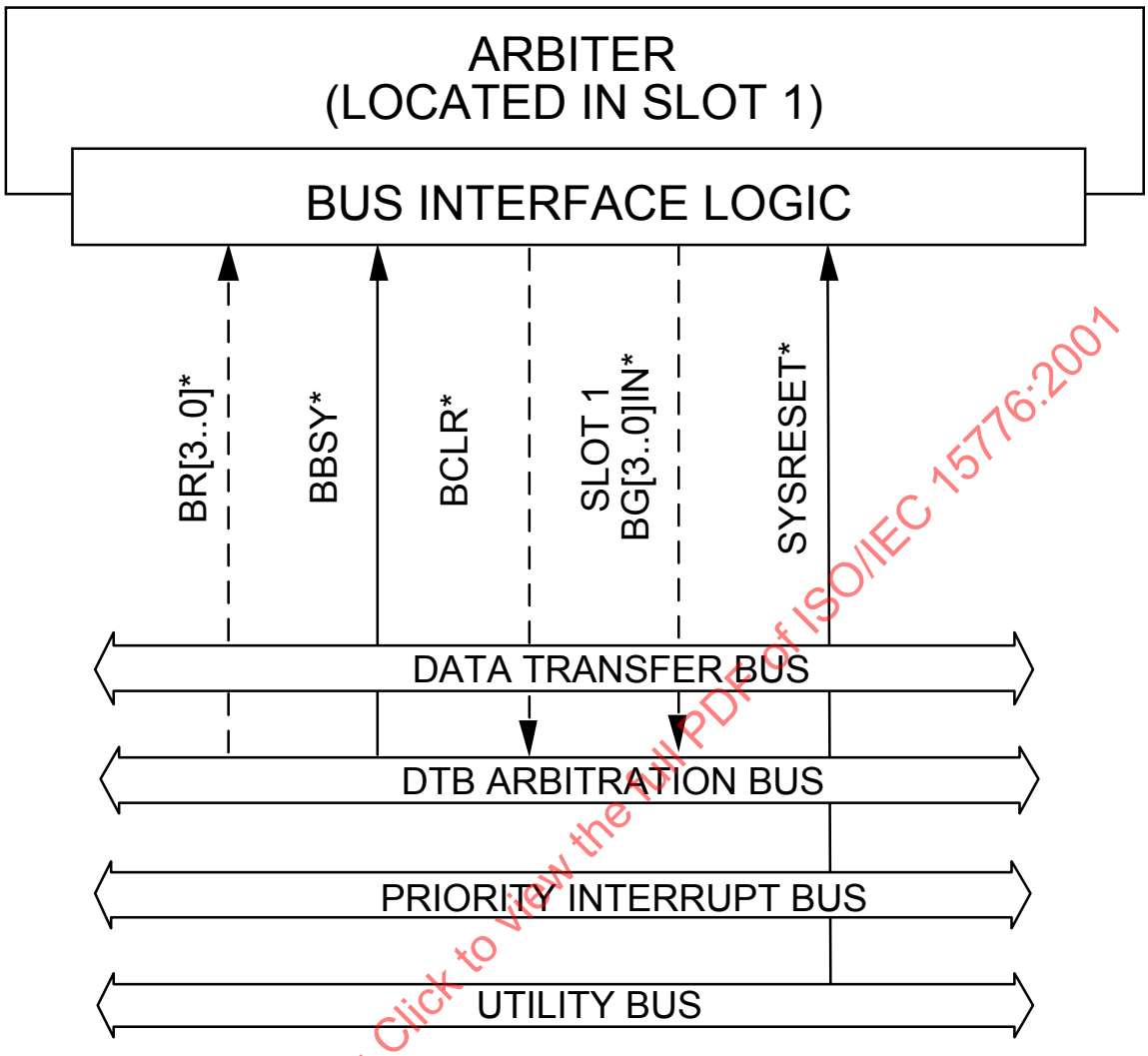
THEN it might grant the bus to a Requester that has since removed its request.

RULE 3.15

The Arbiter **MUST** wait for a minimum of 40 ns after detecting the rising edge of BBSY* before driving any BG[0..3]IN* lines low.

OBSERVATION 3.19

Rule 3.15 guarantees a 40 ns minimum high time on BBSY* (30 ns on the non-System Controller boards) to support future protocols that need to detect the negation of BBSY*.



NOTE The RULEs and PERMISSIONs for monitoring and driving the dotted lines are given in Table 33.

Figure 41 – Block diagram – Arbiter

Table 33 – RULEs and PERMISSIONs specifying the use of dotted lines by the various types of arbiters defined in this standard

Type of arbiter	Use of dotted line
SGL	**MUST** drive Slot 1 BG3IN*(see note). **MUST** ensure Slot 1 BG[2..0]IN* are high (see note). **MUST** monitor BR3*. **MAY** drive BCLR*, or Slot 1 BG[2..0]IN* (see note). **MAY** monitor BR[2..0]*.
RRS	**MUST** drive Slot 1 BG[3..0]IN* (see note). MUST monitor BR[3..0]*. **MAY** drive BCLR*.
PRI	**MUST** drive BCLR* and Slot1 BG[3..0]IN* (see note). **MUST** monitor BR[3..0]*.
NOTE If there is no Requester on the board then as per Rule 3.1 the board propagates the daisy chain signals BG[3..0]IN to BG[3..0]OUT.	

3.3.2 Requester

Each Requester in the system

a) monitors the DEVICE WANTS BUS signal from its on-board Master or Interrupt Handler and generates a bus request when the DTB is needed.

b) if it detects a low level on its BG[3..0]IN* line, and its on-board Master or Interrupt Handler does not need the DTB, then it passes on that low level to its BG[3..0]OUT*.

c) if it detects a low level on its BG[3..0]IN* line, and its on-board Master or Interrupt Handler needs the DTB, it generates an on-board DEVICE GRANTED BUS signal to indicate the DTB is available, and drives the BBSY* signal low.

Three types of Requesters are described in this specification: a Release When Done (RWD) Requester, a Release On Request (ROR) Requester and a FAIR Requester.

The RWD Requester releases BBSY* when the Master or Interrupt Handler drives the on-board DEVICE WANTS BUS signal false.

The ROR Requester does not release BBSY* when its on-board DEVICE WANTS BUS signal goes false unless some other Requester on the bus drives one of the bus request lines low. It monitors the four bus request lines and releases BBSY* only if another bus request is pending. ROR Requesters reduce the number of arbitrations initiated by a Master which is generating a large percentage of the bus traffic. See Figure 42.

FAIR Requesters are used to ensure guaranteed access to the bus when there are more than four Masters and Interrupt Handlers. Fair Requesters do not request the bus if there are other bus requests pending on the same level. When the current "batch" of requests are serviced, other requesters needing access to the bus issue requests. This approach guarantees each requester on a level will get access to the bus. It does not however guarantee that each requester gets equal bandwidth or that the accesses are granted in a consistent order. These are implementation and system dependent.

OBSERVATION 3.17

When configuring a system to provide fairness for more than four Masters or Interrupt Handlers all the requesters in the system should be Fair Requesters. If one or more of the Requesters are not FAIR Requesters, then fairness can still be guaranteed if the sum total of bus traffic generated by the non-Fair Requesters does not exceed the capacity of the bus.

OBSERVATION 3.18

The RWD and ROR capabilities describe under what conditions a requester relinquishes control of the DTB. On the other hand, the FAIR capability describes under what conditions the requester requests control of DTB. Therefore, both the RWD and the ROR requesters can include the FAIR capability as well.

Assuming that the Requester's DEVICE WANTS BUS input is true, when it receives a bus grant it does three things:

a) it drives BBSY* to low;

b) it releases its low on BR[3..0]*;

c) it drives the DEVICE GRANTED BUS on-board signal true, allowing the Master or Interrupt Handler to initiate bus transfers.

These events might occur in any order. It is even possible, although meaningless, that the Master or Interrupt Handler might not use the bus in response to this particular grant.

However, the following RULES apply:

RULE 3.7

On detecting BGxIN going low, the Requester **MUST** drive BBSY* to low for at least 90 ns.

RULE 3.8

On detecting BGxIN going low, the Requester **MUST** release Bus Request BR[3..0]* to high.

RULE 3.9

On detecting BGxIN going low, the Requester **MUST** maintain BBSY* low for at least 30 ns after it releases its bus request.

OBSERVATION 3.8

The 30 nanosecond delay between the bus request's rising edge and the BBSY* rising edge ensures that the Arbiter does not mistakenly interpret the old bus request as a new one and issue another grant.

RULE 3.10

The Requester **MUST** hold BBSY* low until its bus grant goes high.

OBSERVATION 3.9

RULE 3.10 ensures that the BBSY* transition to low has been seen by the Arbiter and that all segments of the bus grant daisy-chain have returned to high, in preparation for the next arbitration.

PERMISSION 3.5

IF a Requester has a bus request pending and, if it sees some other Requester drive BBSY* low

THEN it MAY withdraw its request by releasing the bus request line.

RULE 3.11

IF a Requester withdraws a bus request without having first been granted the bus,

THEN it **MUST** wait to do so until BBSY* goes low and it **MUST** do so within 50 ns after BBSY* goes low.

SUGGESTION 3.2

Design Requesters in such a manner that they pass on the bus grant daisy-chain as fast as possible after receipt of a bus grant. This will improve system performance.

RULE 3.14

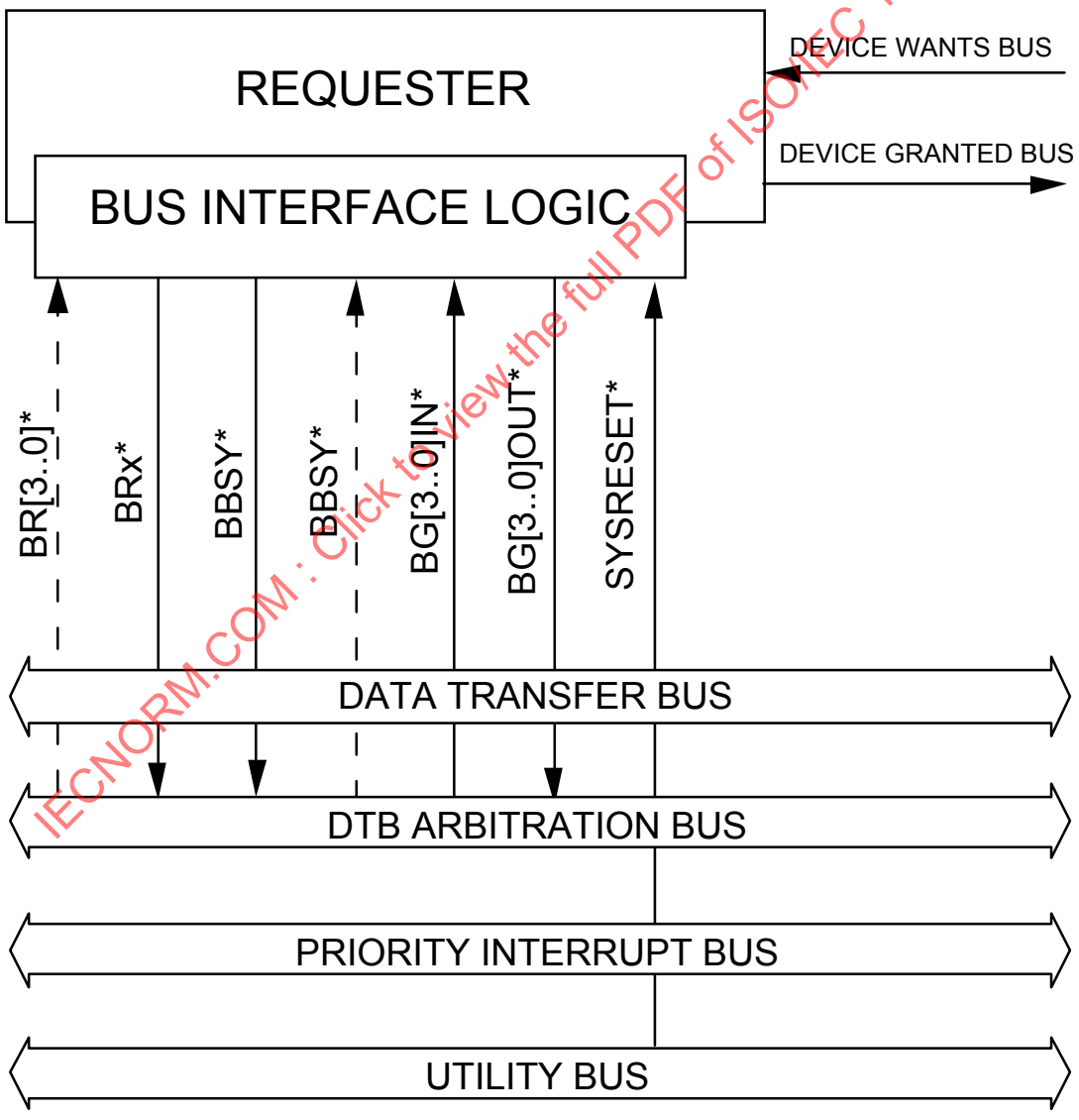
A FAIR Requester **MUST NOT** issue a bus request unless it detects a high on its bus request line.

RECOMMENDATION 3.3

A FAIR Requester should wait at least 40 ns after it detects a high on its bus request line before asserting its bus request.

OBSERVATION 3.20

If a FAIR Requester asserts its request line immediately after detecting the line going high then other requesters may not recognize the line being high. This may result in requesters not receiving fair access to the bus.



NOTE The RULES and PERMISSIONS for monitoring the dotted lines are given in Table 34.

Figure 42 – Block diagram – Requester

Table 34 – RULEs and PERMISSIONs specifying the use of the dotted lines by the various types of requesters defined in this document

Type of arbiter	Use of dotted line
RWD	**MAY** monitor BR[3..0]*. **MAY** monitor BBSY*
ROR	**MUST** monitor BR[3..0]*. **MAY** monitor BBSY*
FAIR	**MUST** monitor the bus request line it is using **MAY** monitor BBSY*

3.3.3 Data transfer bus master

3.3.3.1 Release of the DTB

The bus arbitration protocol determines how and when the DTB is granted to the various Masters and Interrupt Handlers in the system. It does not, however, control when Masters and Interrupt Handlers release the DTB.

Masters and Interrupt Handlers use several criteria in deciding when to release the DTB. Interrupt Handlers give up the bus after their interrupt acknowledge cycle. Masters give up the bus when they finish their data transfers.

Some Masters also monitor the ACFAIL* and BCLR* VMEbus signals. Both of these signals inform the Master that the DTB is needed for some higher priority activity. In the case of BCLR*, the Master's design determines how long it takes to release the bus. For example, a Master on a disk controller board might not be able to relinquish the bus during a disk sector transfer without loss of data, so it might keep the bus until the sector transfer is finished. ACFAIL* informs the Master that an AC power loss has been detected, and the problems the Master will face in surrendering the bus are insignificant compared to the needs of the total system.

RECOMMENDATION 3.1

Design Masters to release the DTB within 200 µs after ACFAIL* goes low, except to participate in the ensuing power failure activities.

OBSERVATION 3.10

The 200 µs time specified in RECOMMENDATION 3.1 is intended to provide enough time for an orderly shut-down of the system.

Whatever criteria are used to decide when to release the DTB, arbitration is done before some other Master or Interrupt Handler begins using it. This arbitration takes place either during the last data transfer or after that transfer, depending on how much notice the Master or Interrupt Handler gives to its on-board Requester.

PERMISSION 3.6

Masters and Interrupt Handlers MAY release the DTB either during or after their last data transfer.

For example, if the Master notifies its on-board Requester that it no longer wants the bus during its last data transfer, the Requester releases BBSY* and arbitration takes place during the last transfer. But if the Master waits until the last transfer has completed before signaling its on-board Requester, the DTB remains idle while the arbitration is done. (This was illustrated in 2.5.)

Clauses 2 and 4 contain RULEs that pertain to the release of the DTB.

SUGGESTION 3.3

Design block transfer Master boards in such a way that they signal their Requester to release BBSY* during the last data strobe of the block transfer. If it is released at the beginning of the block transfer, high priority bus requests initiated during the block transfer might not be taken into account by the Arbiter until the next arbitration cycle.

3.3.3.2 Acquisition of the DTB

To ensure that no DTB line is ever driven to opposite states by two Masters or Interrupt Handlers, when these modules take control of the DTB they are constrained by the following rule.

RULE 3.12

When a Master or Interrupt Handler is given control of the DTB by its on-board Requester, it **MUST** wait until it detects AS* high before turning on its DTB drivers.

OBSERVATION 3.11

IF the previous Master or Interrupt Handler releases the bus DURING its last data transfer,

THEN RULE 3.12 ensures that the data transfer will be finished before the new Master or Interrupt Handler starts using the DTB. (If the previous Master or Interrupt Handler waited until the data transfer was finished before releasing the bus, AS* would already be high.)

3.3.3.3 Other information

RECOMMENDATION 3.2

To allow for prompt servicing of interrupt requests and for optimum use of the DTB, design Masters in such a way that they release the DTB as soon as possible after they detect BCLR* low.

PERMISSION 3.7

A Master or Interrupt Handler MAY have more than one Requester, where each Requester generates bus requests on a different bus request line.

OBSERVATION 3.12

Where a Master or Interrupt Handler has two or more Requesters, it can do high priority data transfers using one Requester and low priority transfers using another Requester.

3.4 Typical operation

3.4.1 Arbitration of two different levels of bus request

Figures 43 and 44 illustrate the sequence of events that takes place when two Requesters send simultaneous bus requests to a PRI Arbiter on different bus request lines. When the sequence begins, each of the Requesters drives its respective bus request line low (Requester A drives BR1* and Requester B drives BR2*). The Arbiter detects BR1* and BR2* low simultaneously, and it drives BG2IN* low to its own slot (Slot 1). That BG2IN* signal is monitored by Requester B (also in Slot 1). When Requester B detects BG2IN* low, it responds by driving BBSY* low. Requester B then releases the BR2* line and informs its own Master (Master B) that the DTB is available. When BBSY* goes low, the Arbiter drives BG2IN* of Slot 1 high.

When Master B completes its data transfer(s), and signals that fact by driving DEVICE WANTS BUS false, Requester B releases BBSY*, provided that its BG2IN* has been received high and 30 ns have elapsed since it released BR2*.

The Arbiter interprets the release of BBSY* as a signal to arbitrate any current bus requests. Since BR1* is the only bus request being driven low, the Arbiter grants the DTB to Requester A by driving BG1IN* low. Requester A responds by driving BBSY* low. When Master A completes its data transfer(s), and signals that fact by driving DEVICE WANTS BUS false, Requester A releases BBSY*, provided that its BG1IN* has been received high and 30 ns have elapsed since it released BR1*.

In this example, since no bus request lines are low when Requester A releases BBSY*, the Arbiter waits until it detects a bus request.

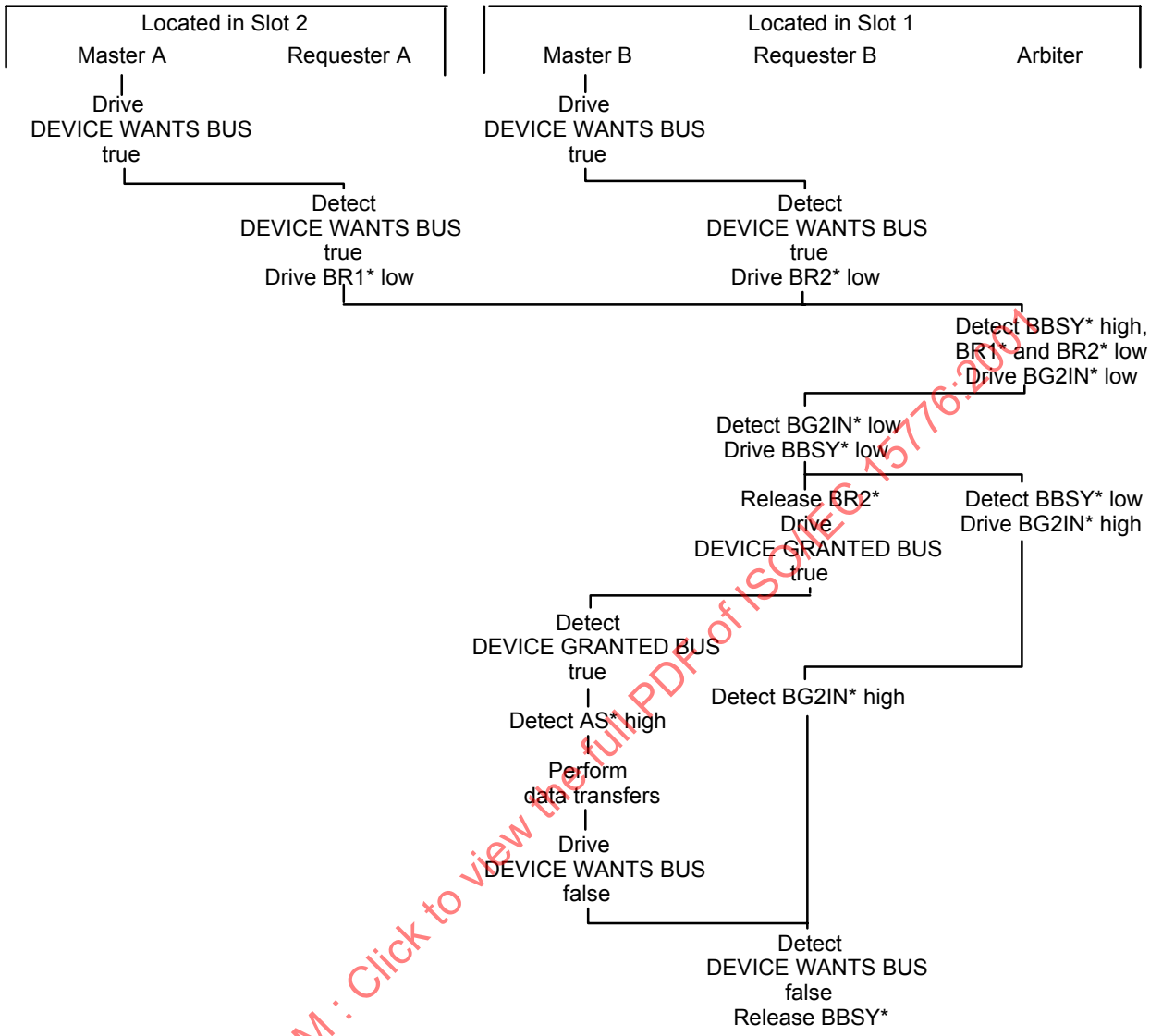
OBSERVATION 3.13

The description illustrated in Figures 43 and 44 would hold for both PRI and RRS Arbiters, unless we consider an RRS Arbiter where the last active request was level BR2*. In this case, the Arbiter would process the BR1* request first and then proceed to the BR2* request.

OBSERVATION 3.14

BBSY* and the bus grants are fully interlocked as shown in Figure 44.

1) The Arbiter does not drive the bus grant high until it detects BBSY* low.
2) The Requester does not release BBSY* to high until it detects the bus grant high.
3) The Arbiter does not drive the next bus grant low until it detects BBSY* high for 40 ns.
4) The next Requester does not drive BBSY* low until it detects the bus grant low.



NOTE DEVICE WANTS BUS and DEVICE GRANTED BUS are on-board signals between the Master and its Requester (see Figure 42).

Figure 43 – Arbitration flow diagram two requesters, two request levels

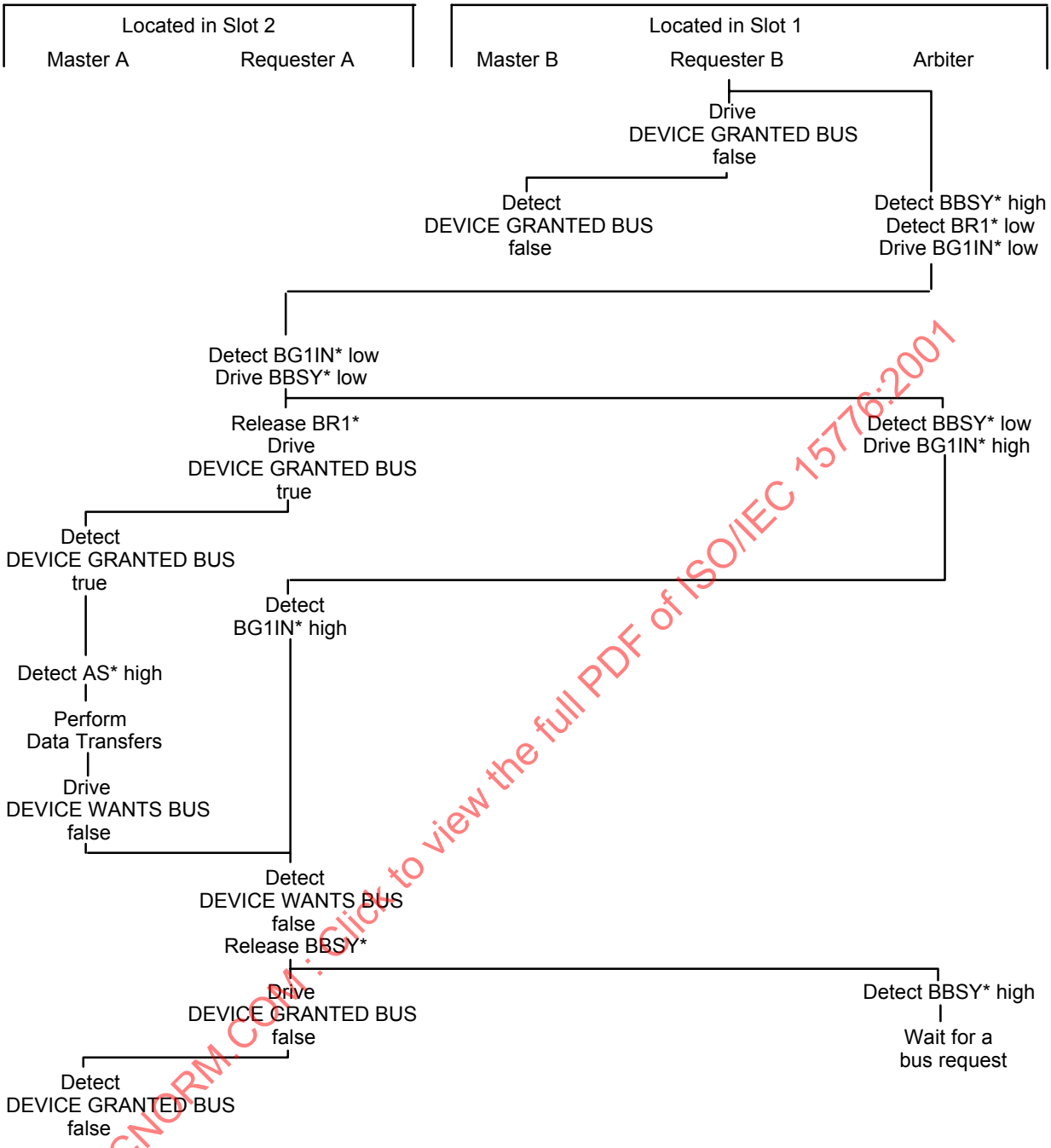
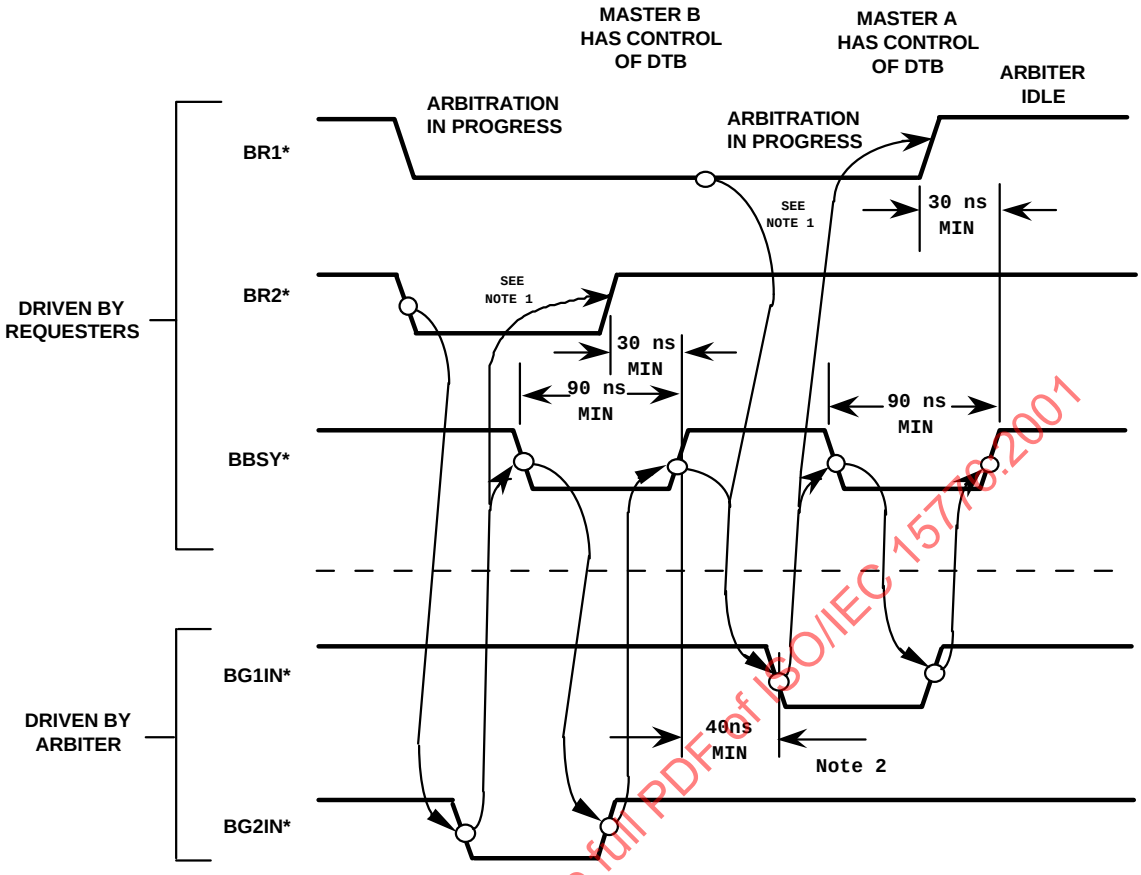


Figure 43 – Arbitration flow diagram two requesters, two request levels (*continued*)



NOTE 1 In this example each Requester maintains its bus request line low until it is granted the DTB. In some cases a Requester might release its bus request line without receiving a bus grant (see 3.3.2).

NOTE 2 This specification has been added to accommodate systems that require a minimum BBSY* high time such as VME64 LOCK protocols and Autobahn.

Figure 44 – Arbitration sequence diagram two requesters, two request levels

3.4.2 Arbitration of two bus requests on the same bus request line

Figures 45 and 46 illustrate the sequence of events which take place when an ROR Requester and an RWD Requester send simultaneous requests to a PRI Arbiter on a common bus request line. In this example, the Arbiter and RWD Requester are located on the system controller board in Slot 1, with the ROR Requester located in Slot 2. When the sequence begins, both of the Requesters drive BR1* low simultaneously. The Arbiter then drives BG1IN* low to its own slot (Slot 1). That BG1IN* signal is monitored by Requester A (also in Slot 1). When Requester A detects BG1IN* low, it responds by driving BBSY* low. Requester A then releases BR1* and informs Master A that the DTB is available.

OBSERVATION 3.15

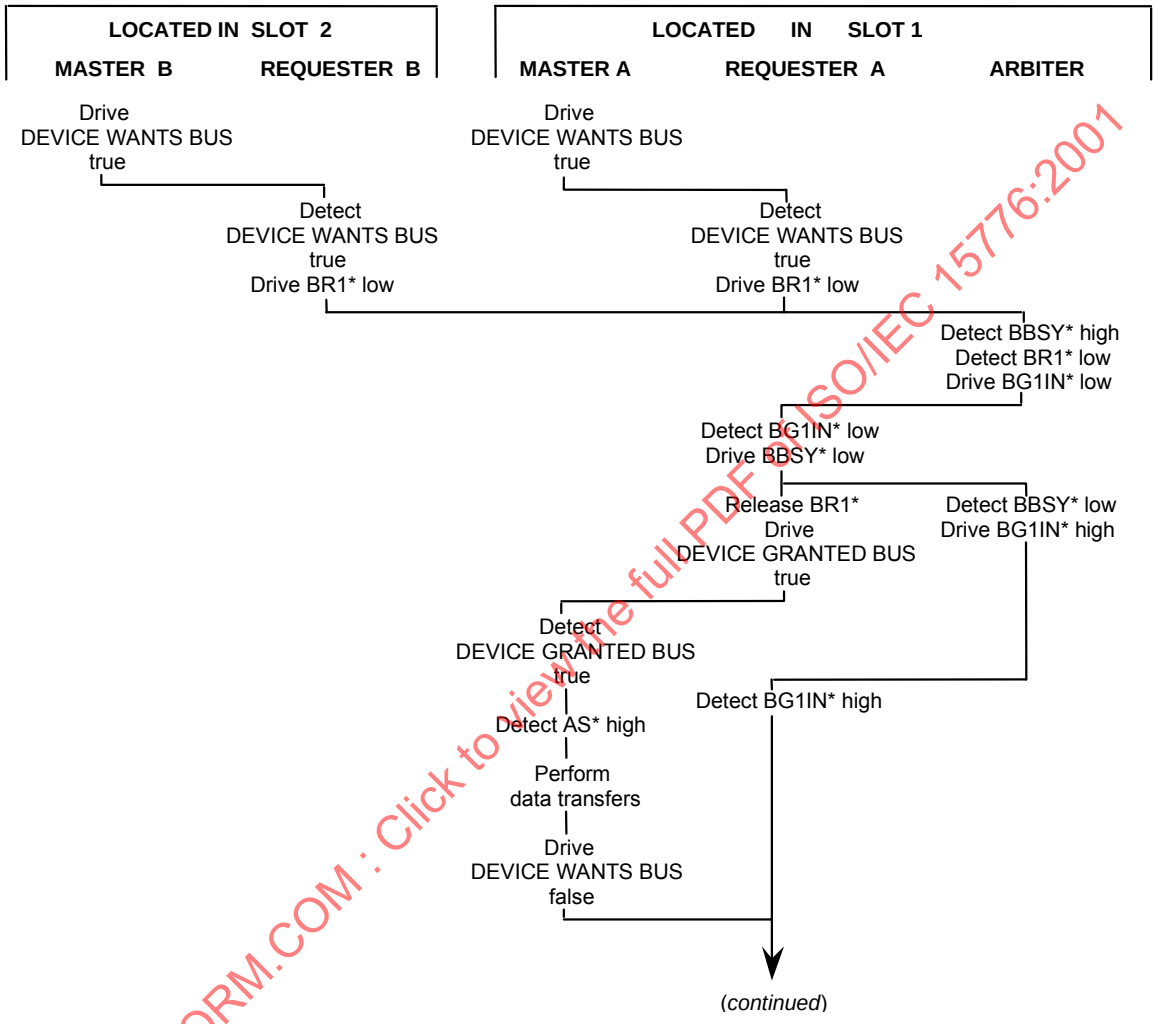
Even though Requester A releases BR1*, Requester B continues to drive it low (see Figures 45 and 46).

After detecting BBSY* low, the Arbiter drives BG1IN* high. When Master A has completed its data transfer(s), it drives DEVICE WANTS BUS false. When Requester A detects this, and when the 30 ns delay since the release of BR1* has been satisfied, Requester A releases BBSY*.

The Arbiter interprets the release of BBSY* as a signal to arbitrate any current bus requests. Since the BR1* line is still low, the Arbiter drives BG1IN* low again. When Requester A detects BG1IN* low, it drives its BG1OUT* low because it does not need the DTB. Requester B then detects the low on its BG1IN* and responds by driving BBSY* low. When the Arbiter detects the low on BBSY*, it drives BG1IN* high, which causes Requester A to drive its BG1OUT* high.

Some time later, when Master B has finished its data transfers, it drives DEVICE WANTS BUS false, indicating that it has finished using the DTB.

Since Requester B is an ROR Requester, it does not release BBSY*, but keeps it driven low. In the event that Master B needs to use the DTB again, no arbitration will be required. In this example, however, Requester A drives BR1* low, indicating a need to use the DTB, and Requester B (which is monitoring the bus request lines) releases the BBSY* line. The Arbiter then grants the DTB to Requester A.



NOTE DEVICE WANTS BUS and DEVICE GRANTED BUS are on-board signals between the Master and its Requester (see Figure 42).

Figure 45 – Arbitration flow diagram two requesters, same request level

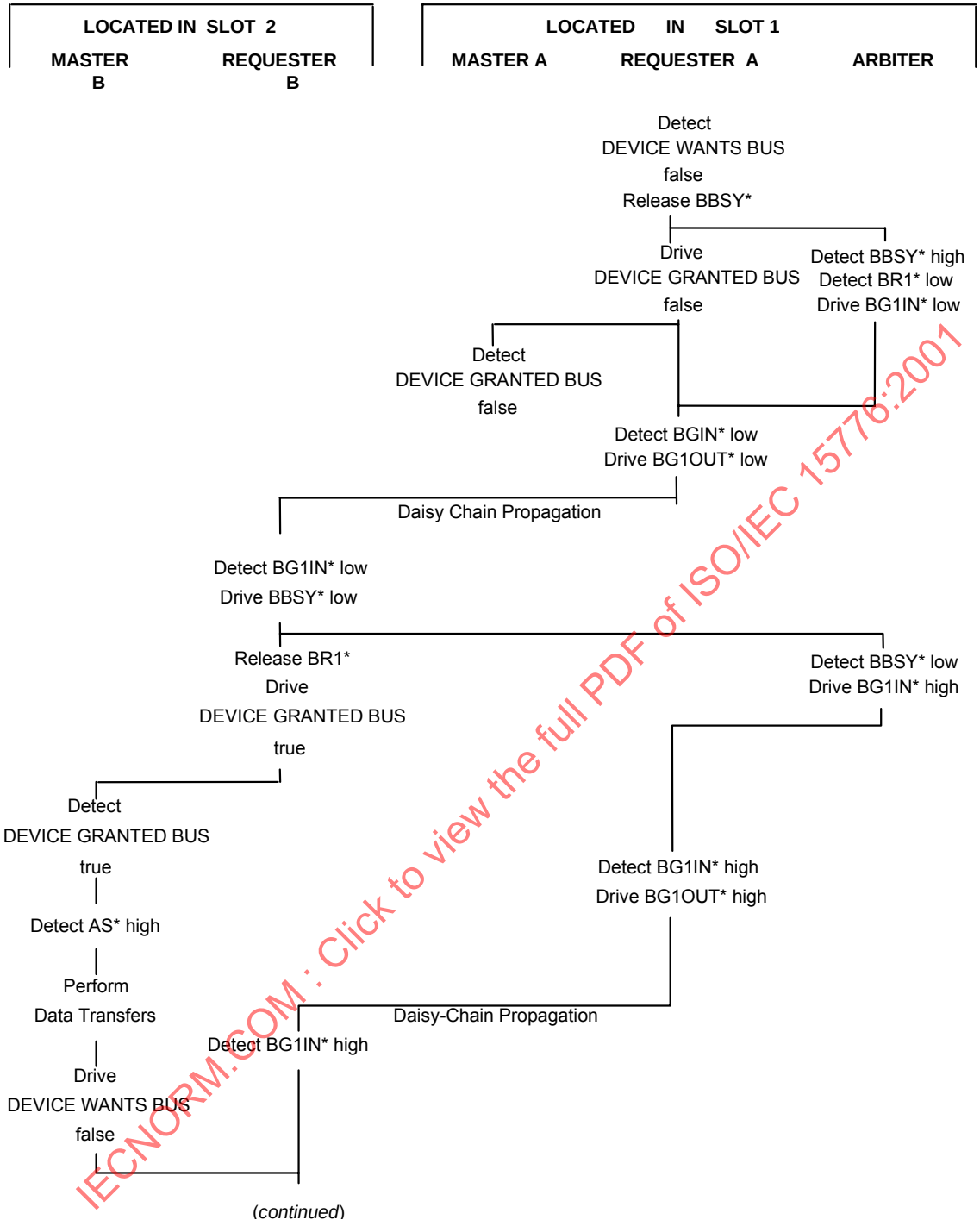


Figure 45 – Arbitration flow diagram two requesters, same request level (*continued*)

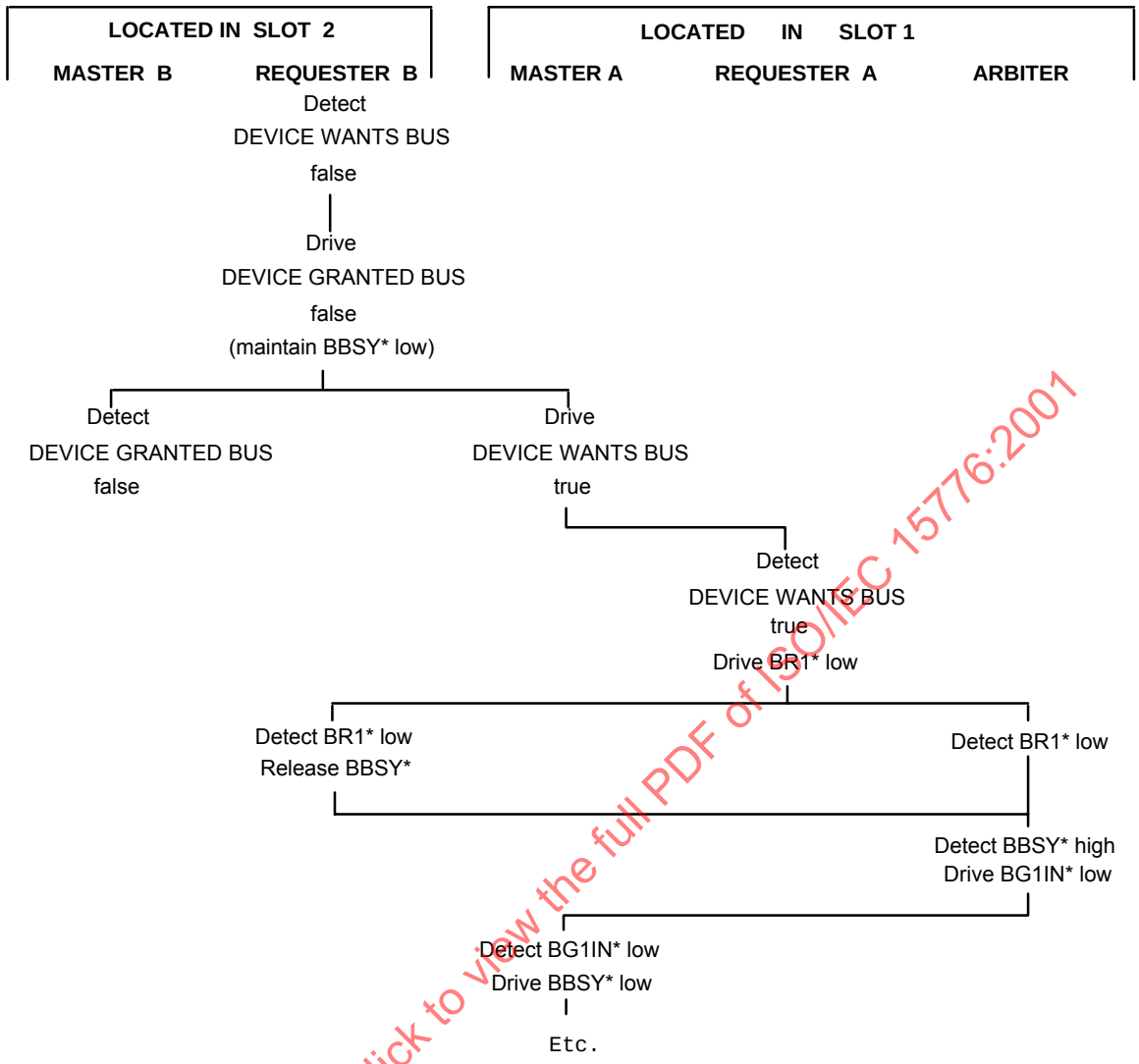
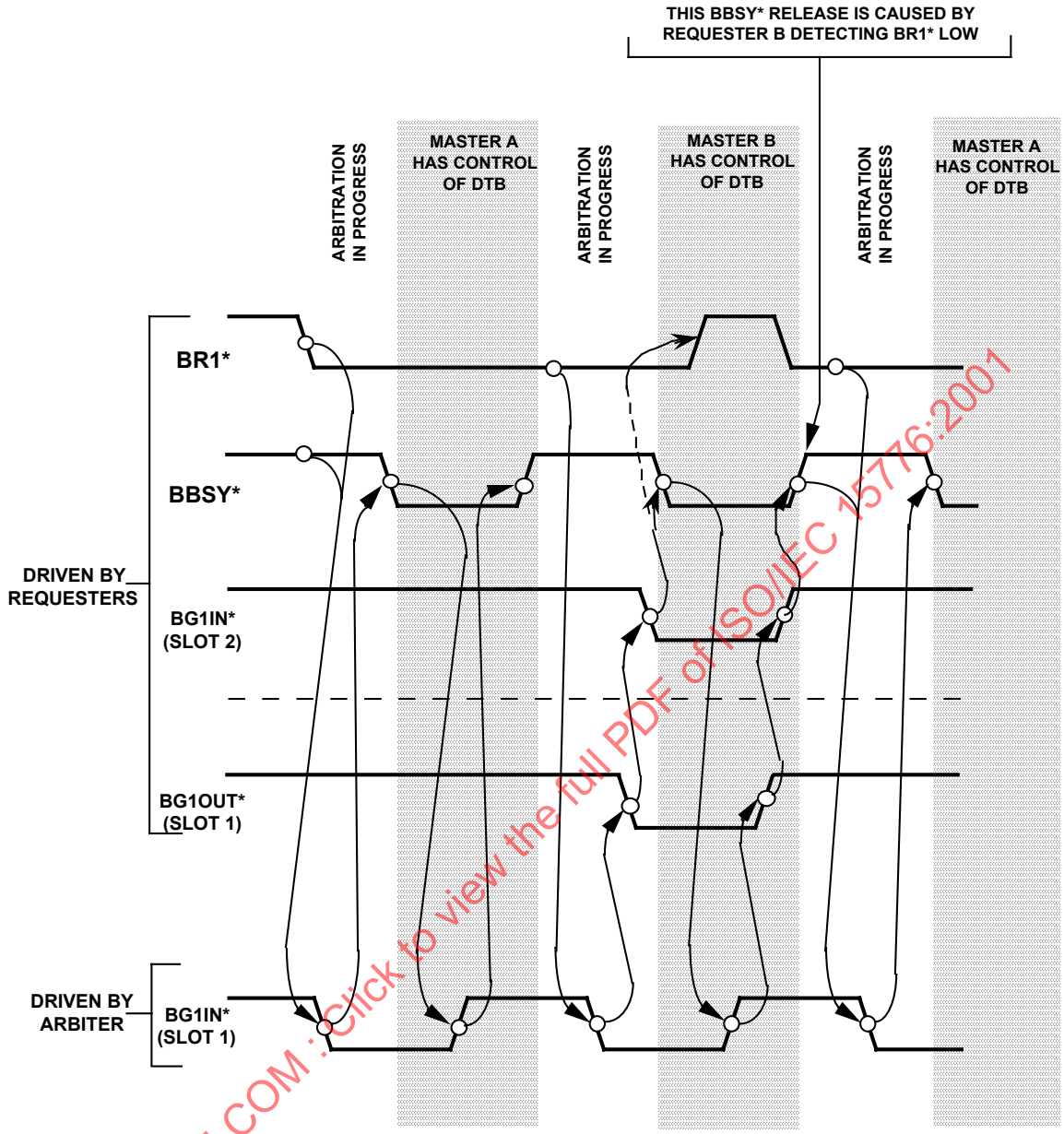


Figure 45 – Arbitration flow diagram two requesters, same request level (*continued*)



NOTE In this example, each Requester maintains its bus request line low until it is granted the DTB. In some cases a Requester might release its bus request line without receiving a bus grant (see 3.3.2).

Figure 46 – Arbitration sequence diagram two requesters, same request level

3.5 Race conditions between master requests and arbiter grants

Suppose that there are two Requesters, Requester A and Requester B, that share a common bus request line. Requester B, which is farther down the daisy-chain, requests the bus and the Arbiter drives the corresponding bus grant line low. This bus grant arrives at Requester A just as Master A signals that it wants the bus. If Requester A is improperly designed, this situation might cause it to momentarily drive its BG[3..0]OUT* line low and then high again resulting in a low-going transient.

RULE 3.13

Requesters **MUST** be designed to ensure that no momentary low-going transients are generated on their BG[3..0]OUT* out line.

OBSERVATION 3.16

If the Requester is designed in such a way that it latches the state of the on-board DEVICE WANTS BUS line upon the falling edge of its bus grant in line, and if that signal is in transition when the falling edge occurs, the outputs of the latch will sometimes oscillate, or remain in the threshold region between the high and low levels, for a short time. Because of this, the VMEbus specification does not set a time limit for the Requester to pass along the bus grant. It only prohibits the Requester from generating a low-going transient on its BG[3..0]OUT* line which might be interpreted as a bus grant by a Requester further down the daisy-chain.

PERMISSION 3.8

IF a Requester detects that its on-board Master needs the bus between the time that it receives a bus grant intended for another Requester and the time it would pass that bus grant on,

THEN it MAY treat the bus grant as its own. In this case the other Requester will maintain its bus request until another bus grant is issued.

4 Priority interrupt bus

4.1 Introduction

The VMEbus includes a Priority Interrupt Bus which provides the signal lines needed to generate and service interrupts. Figure 47 shows a typical VMEbus system. Interrupters use the Priority Interrupt Bus to send interrupt requests to Interrupt Handlers which respond to these requests.

Any system which has interrupt capability includes software routines that are called interrupt service routines, which are invoked by the interrupts. Interrupt subsystems can be classified into two groups:

a) single handler systems, which have only one Interrupt Handler that receives and services all bus interrupts;

b) distributed systems, which have two or more Interrupt Handlers that receive and service bus interrupts.

4.1.1 Single handler systems

In a single handler system, all interrupts are received by one Interrupt Handler and all interrupt service routines are executed by one processor. Figure 48 shows the interrupt structure of a single handler system. This type of architecture is well suited to machine or process control applications, where a supervisory processor coordinates the activities of dedicated processors. The dedicated processors are typically interfaced to the machine or the process being controlled.

The supervisory processor is the destination for all bus interrupts, servicing them in a prioritized manner. The dedicated processors are not required to service interrupts from the bus and can give primary attention to controlling a machine or process.

4.1.2 Distributed systems

Figure 49 shows the interrupt structure of a distributed system. This system includes two or more Interrupt Handlers, each servicing only a subset of the bus interrupts. In a typical implementation, each

of the Interrupt Handlers resides on a different processor board. This type of architecture is well suited to distributed computing applications, where multiple, co-equal processors execute the application software. As each of the co-equal processors executes part of the system software, it might need to communicate with the other processors. In the distributed system, each processor services only those interrupts directed to it, establishing dedicated communication paths among all processors.

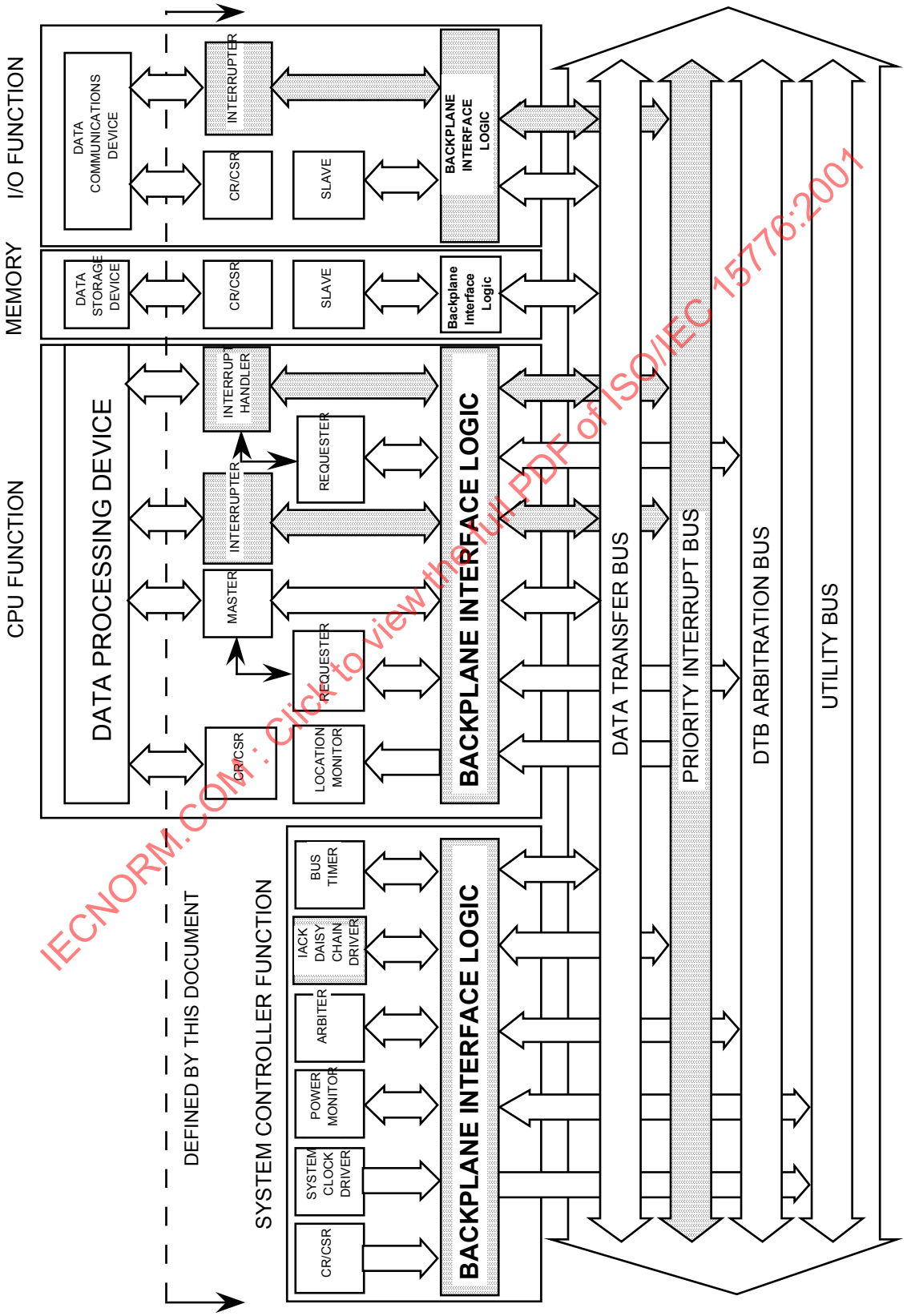


Figure 47 – Priority interrupt bus functional diagram

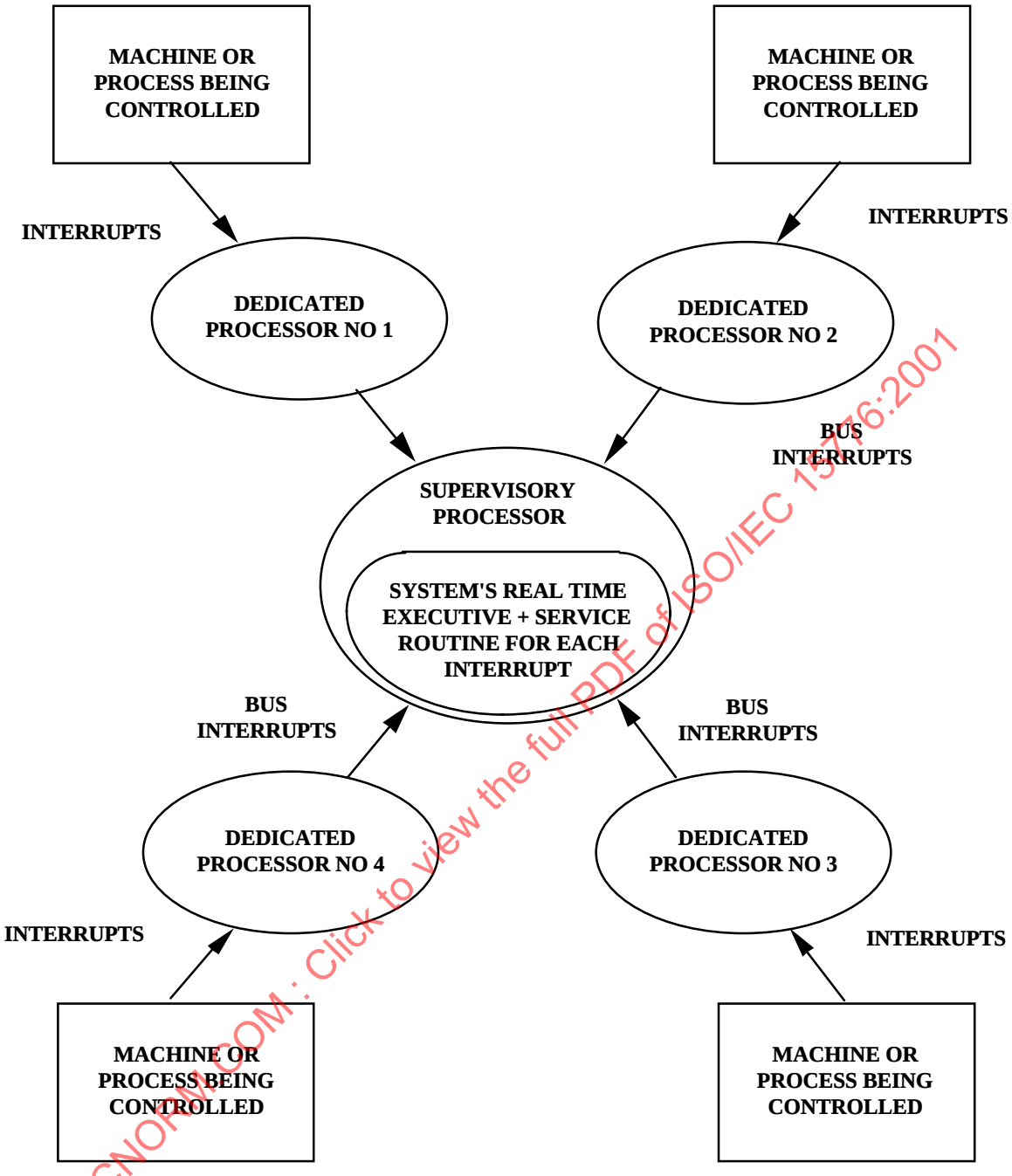


Figure 48 – Interrupt subsystem structure – Single handler system

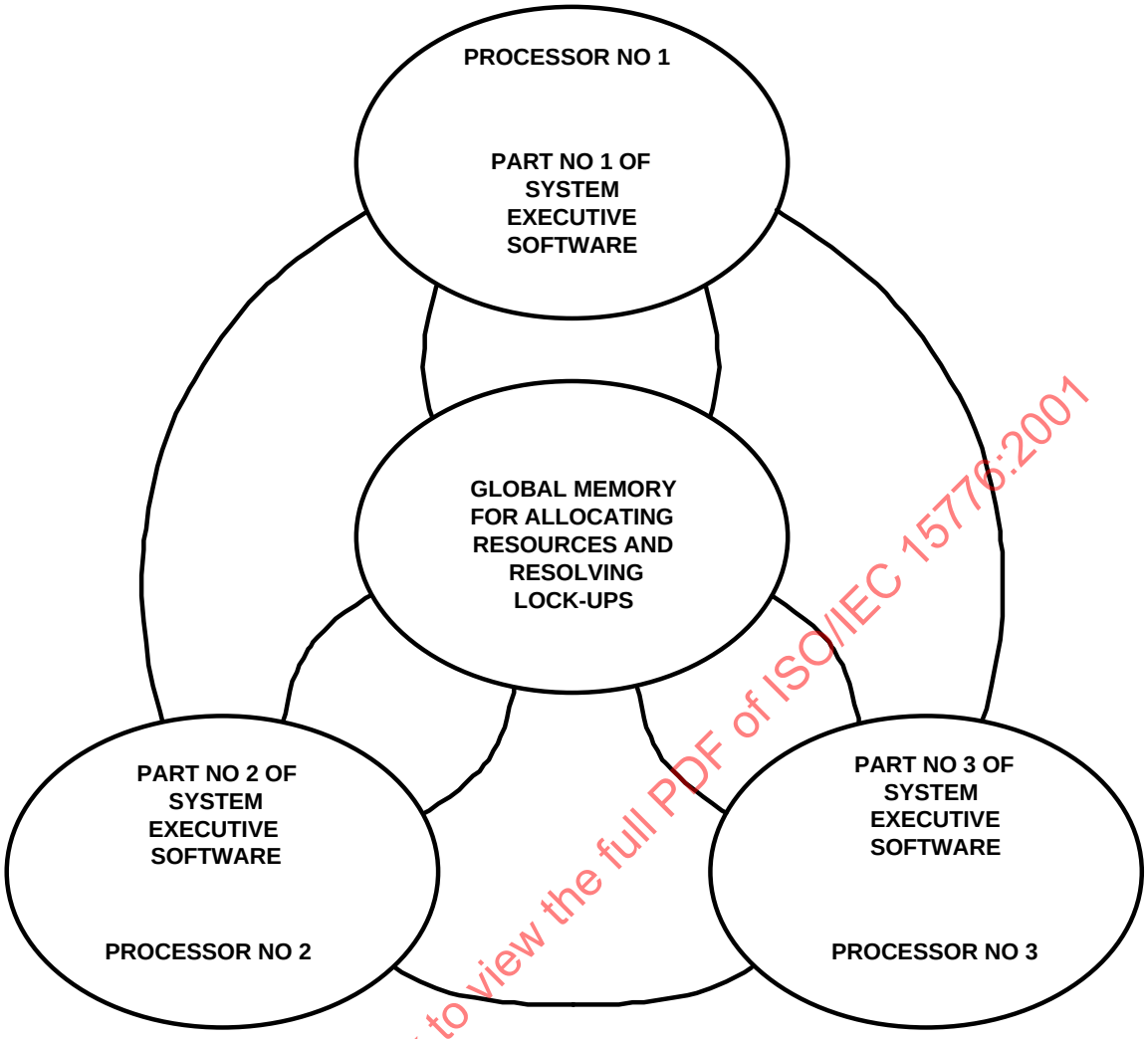


Figure 49 – Interrupt subsystem structure – Distributed system

4.2 Priority interrupt bus lines

The Data Transfer Bus, the Arbitration Bus, and the Priority Interrupt Bus are all used in the process of generating and handling bus interrupts.

The following discussion of the Priority Interrupt Bus assumes that the reader understands the operation of the Data Transfer Bus described in clause 2, and the Arbitration Bus described in clause 3.

The Priority Interrupt Bus consists of seven interrupt request signal lines, one interrupt acknowledge line, and one interrupt acknowledge daisy-chain:

IRQ1*	Interrupt Request 1
IRQ2*	Interrupt Request 2
IRQ3*	Interrupt Request 3
IRQ4*	Interrupt Request 4
IRQ5*	Interrupt Request 5
IRQ6*	Interrupt Request 6
IRQ7*	Interrupt Request 7
IACK*	Interrupt Acknowledge
IACKIN*/IACKOUT*	Interrupt Acknowledge Daisy-Chain

4.2.1 Interrupt request lines

Interrupters request interrupts by driving an interrupt request line low. In a single handler system, these interrupt request lines are prioritized, with IRQ7* having the highest priority.

4.2.2 Interrupt acknowledge line

The IACK* line runs the full length of the backplane and is connected to the IACKIN* pin of Slot 1 (see Figure 50). When driven low, the IACKIN* pin causes the IACK Daisy-Chain Driver, located in Slot 1, to propagate a falling edge down the interrupt acknowledge daisy-chain.

4.2.3 Interrupt acknowledge daisy-chain - IACKIN*/IACKOUT*

Each of the seven interrupt request lines can be shared by two or more Interrupter modules. The interrupt acknowledge daisy-chain assures that only one Interrupter responds to the interrupt acknowledge cycle. This daisy-chain line passes through each board on the VMEbus. Each Interrupter that is driving an interrupt request line low waits for a falling edge to arrive at its IACKIN* daisy-chain input. Only upon receiving this falling edge does an Interrupter respond to an Interrupt Acknowledge Cycle. It does not pass the falling edge down the Daisy-Chain, preventing other Interrupters from responding to the Interrupt Acknowledge Cycle.

RULE 4.1

IF a VMEbus backplane slot is not occupied by a board, and if there are boards farther down the interrupt acknowledge daisy-chain,

THEN jumpers or a functionally equivalent mechanism **MUST** be installed at the empty slot to pass-through the daisy-chain signal.

RULE 4.50

IF a VMEbus board does not generate interrupt requests,

THEN it **MUST** propagate the IACK daisy chain from its IACKIN* line to its IACKOUT* line.

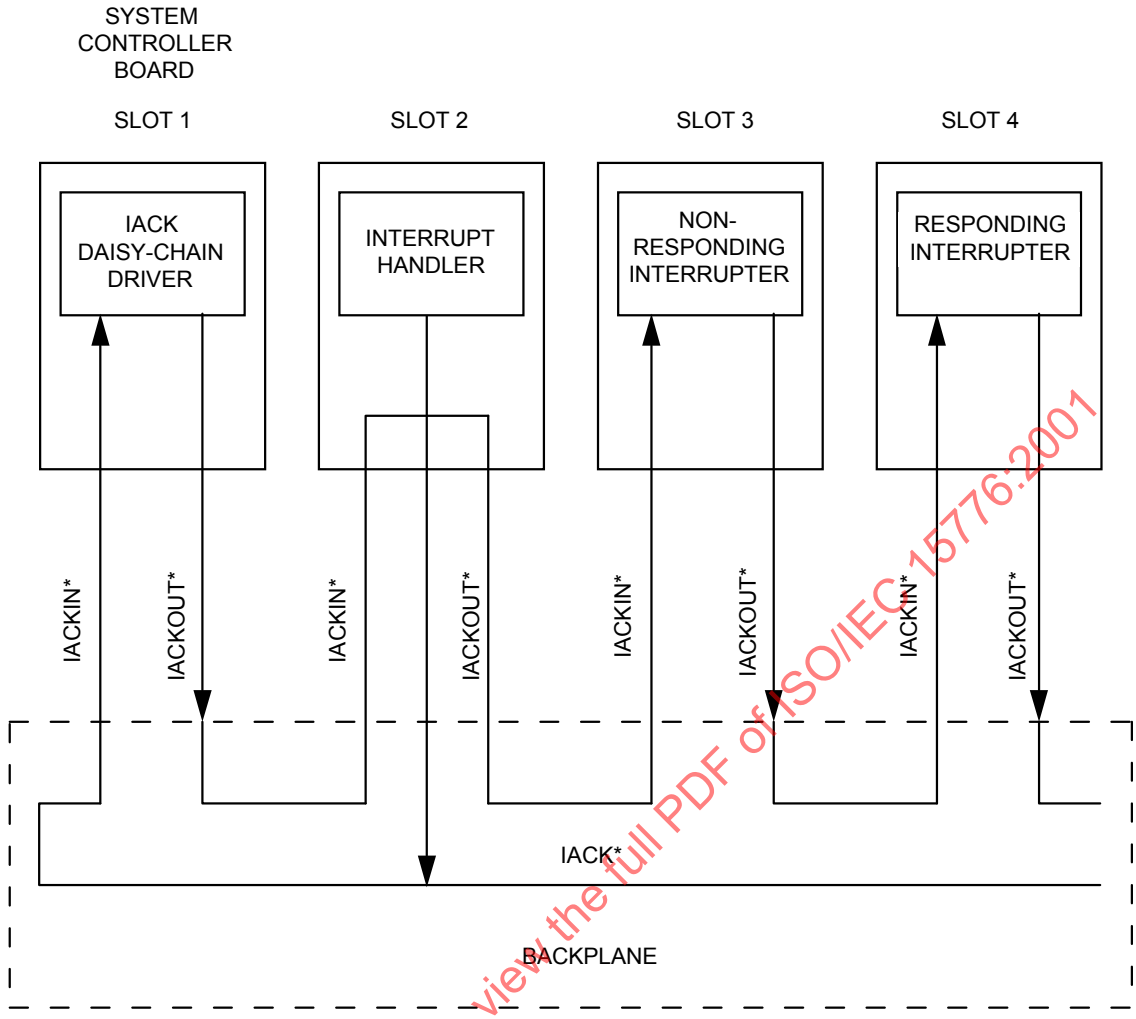


Figure 50 – IACKIN*/IACKOUT* DAISY-CHAIN

4.3 Priority interrupt bus modules – Basic description

There are three types of functional modules associated with the Priority Interrupt Bus: Interrupters, Interrupt Handlers, and an IACK Daisy-Chain Driver. The capabilities of Interrupt Handlers and Interrupters are described by a list of mnemonics that show what interrupt acknowledge cycle types they can generate and accept.

Subclauses 4.3.1 through 4.3.3 provide block diagrams for the three Priority Interrupt Bus modules: Interrupt Handler, Interrupter, and IACK Daisy-Chain Driver.

RULE 4.2

Output signal lines shown with solid lines in Figures 39 through 41 **MUST** be driven by the module, unless it would always drive them high.

OBSERVATION 4.1a

IF a bused output line is not driven,

THEN terminators on the backplane ensure that it is high.

RULE 4.3

Input signal lines shown with solid lines in Figures 51 through 53 **MUST** be monitored and responded to in the appropriate fashion.

OBSERVATION 4.2

RULEs and PERMISSIONs for driving and monitoring signal lines shown with dotted lines in Figures 51 through 53, are given in Tables 35 and 36.

4.3.1 Interrupt handler

The Interrupt Handler is used to accomplish several tasks.

a) It prioritizes the incoming interrupt requests within its assigned group of interrupt request lines (highest of IRQ[7..1]*).

b) It uses its on-board Requester to request the DTB and, when granted use of the DTB, initiates an interrupt acknowledge cycle, reading a Status/ID from the Interrupter being acknowledged.

c) It initiates the appropriate interrupt servicing sequence based on the information received in the Status/ID.

OBSERVATION 4.3

The VMEbus specification does not dictate what will happen during the interrupt servicing sequence. Servicing of the interrupt might or might not involve use of the VMEbus.

The Interrupt Handler uses the DTB to read a Status/ID from the Interrupter. In this respect, the Interrupt Handler acts like a Master and the Interrupter acts like a Slave. However, there are four important differences. The Interrupt Handler:

a) always drives IACK* low;

b) is not required to drive the address modifier lines;

c) only uses the lowest three address lines A[3..1];

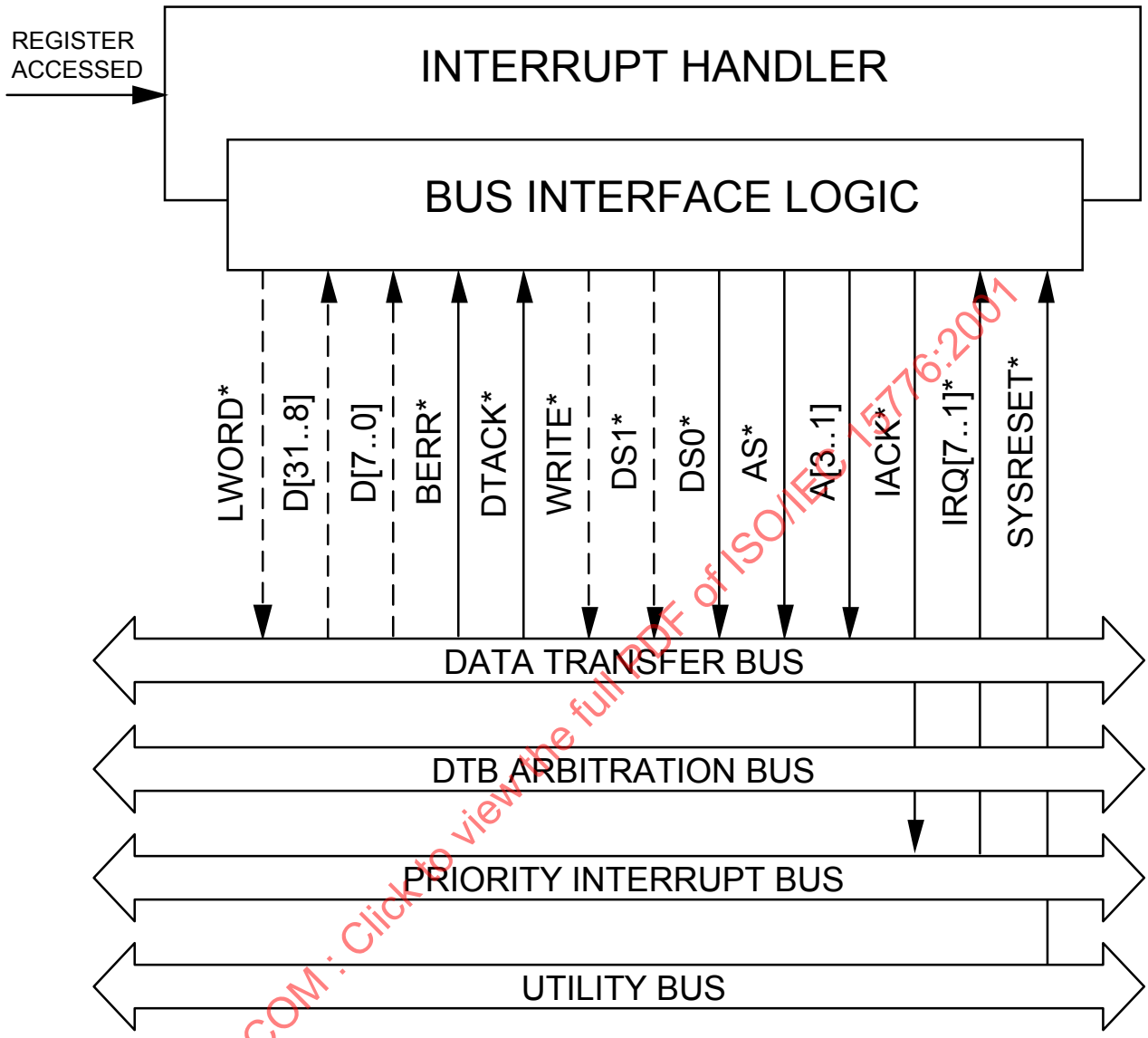
d) never drives the data bus.

The Interrupt Handler always drives IACK* low when it accesses the bus. The Master either drives it high or does not drive it at all.

The Interrupt Handler does not have to drive the address modifier lines with a valid code and it only drives the lowest three address lines A[3..1]. with valid information. The levels of these three address lines indicate which of the seven interrupt request lines is being acknowledged, as shown in Table 41. A Master drives the address bus with the address of the Slave being accessed and provides an address modifier code on the address modifier lines.

The Interrupt Handler does not drive the data lines (i.e., it does not “write” to the Interrupter) and does not have to drive the WRITE* line. A Master uses the data lines to a Slave bidirectionally and, during normal use, drives WRITE* low or high as required.

A block diagram of the Interrupt Handler is shown in Figure 51.



NOTE The RULEs and PERMISSIONs for monitoring and driving the dotted lines are given in Table 35.

Figure 51 – Block diagram – Interrupt handler

Table 35 – RULEs and PERMISSIONs specifying the use of the dotted lines in Figure 51 by the various types of interrupt handlers

Type of Interrupt Handler	Use of dotted lines
D08(O)	**MUST** monitor D[7..O]. **MAY** drive LWORD* and DS1*. **MAY** monitor D[31..0].
D16	**MUST** drive DS1*. **MUST** monitor D[15..0]. **MAY** drive LWORD*. **MAY** monitor D[31..16].
D32	**MUST** drive DS1* and LWORD*. **MUST** monitor D[31..0].
ALL	**MUST NOT** drive WRITE* low.
NOTE The mnemonics D08(O), D16, and D32 are defined in Table 39.	

4.3.2 Interrupter

The Interrupter functions as follows.

a) It requests an interrupt from the Interrupt Handler which monitors its interrupt request line.

b) **IF** it receives a falling edge on the interrupt acknowledge daisy-chain input,

THEN

IF it is requesting an interrupt **AND** the levels on the three valid address lines correspond to the interrupt request line it is using, **AND** the width of the requested Status/ID is either equal to, or greater than the size it can supply,

THEN it supplies a Status/ID,

ELSE it passes the falling edge down the interrupt acknowledge daisy-chain.

Each Interrupter module drives only one interrupt request line. The VMEbus specification describes a board that generates interrupt requests on several interrupt lines as having several Interrupter modules.

PERMISSION 4.1

Since the Interrupter is just a conceptual model, logic on a VMEbus board MAY be shared between several Interrupter modules.

The Interrupter uses one of seven lines to request an interrupt. It then monitors the lowest three lines of the address bus A[3..1], the IACKIN*/IACKOUT* daisy-chain, and optionally IACK*, to determine when its interrupt is being acknowledged. When acknowledged, it places its Status/ID on the data bus and signals the Interrupt Handler that the Status/ID is valid by driving DTACK* low.

There are five primary differences in the use of the DTB by the Interrupter and the Slave. The Interrupter:

a) only responds when its IACKIN* is low;

b) does not have to monitor the address modifier lines;

c) only monitors the lowest three address lines;

d) does not monitor the WRITE* line;

e) is permitted to respond with data of a different size than that requested.

The Slave monitors AS*, and interprets a falling edge on AS* as the signal that a valid bus cycle is in progress. It then proceeds to decode the appropriate number of address lines and the address modifier lines and then, based on this information, it determines whether it was addressed. However, the Slave responds only if IACK* is high.

The Interrupter, on the other hand, interprets the falling edge on its IACKIN* line as a signal that it can respond to the interrupt acknowledge cycle in progress. It decodes only the lowest three address lines A[3..1], ignoring the address modifier lines.

The Interrupter does not need to monitor WRITE*, since it is never written to. Slaves need to monitor WRITE* so that they can distinguish read cycles from write cycles.

The Interrupter places a Status/ID on the bus and responds with DTACK* even if the LWORD*, DS1*, and DS0* lines call for a Status/ID whose width is greater than the Interrupter is able to provide. For example, the Interrupt Handler might drive LWORD* and both data strobes low, indicating that it will read 32-bits of Status/ID from D[31..0], but a D08(O) Interrupter would still respond with its 8-bit Status/ID on D[7..0]. In contrast, when a Slave cannot provide the requested data width, it either responds with BERR* or does not respond at all, typically resulting in a bus time-out.

OBSERVATION 4.4

When an Interrupter places a Status/ID on the data bus, any undriven data lines are read by the Interrupt Handler as high because of the bus terminators. For example, if a D16 Interrupt Handler initiates a double byte interrupt acknowledge cycle, a D08(O) Interrupter would place an 8-bit Status/ID on D[7..0]. The upper 8 bits, read by the Interrupt Handler from D[15..8], are read as ones (high), since they are not driven by the D08(O) Interrupter.

RULE 4.4

Before responding to an interrupt acknowledge cycle, the Interrupter:

1 **MUST** have an interrupt request pending;

2 the level of that request **MUST** match the level indicated on A[3..1];

3 the width of the requested Status/ID **MUST** be equal to or greater than the size it can respond with;

4 it **MUST** have received an incoming falling edge on its IACKIN* daisy-chain input.

IF any of the four conditions above are not met

THEN the Interrupter **MUST NOT** respond to the interrupt acknowledge cycle.

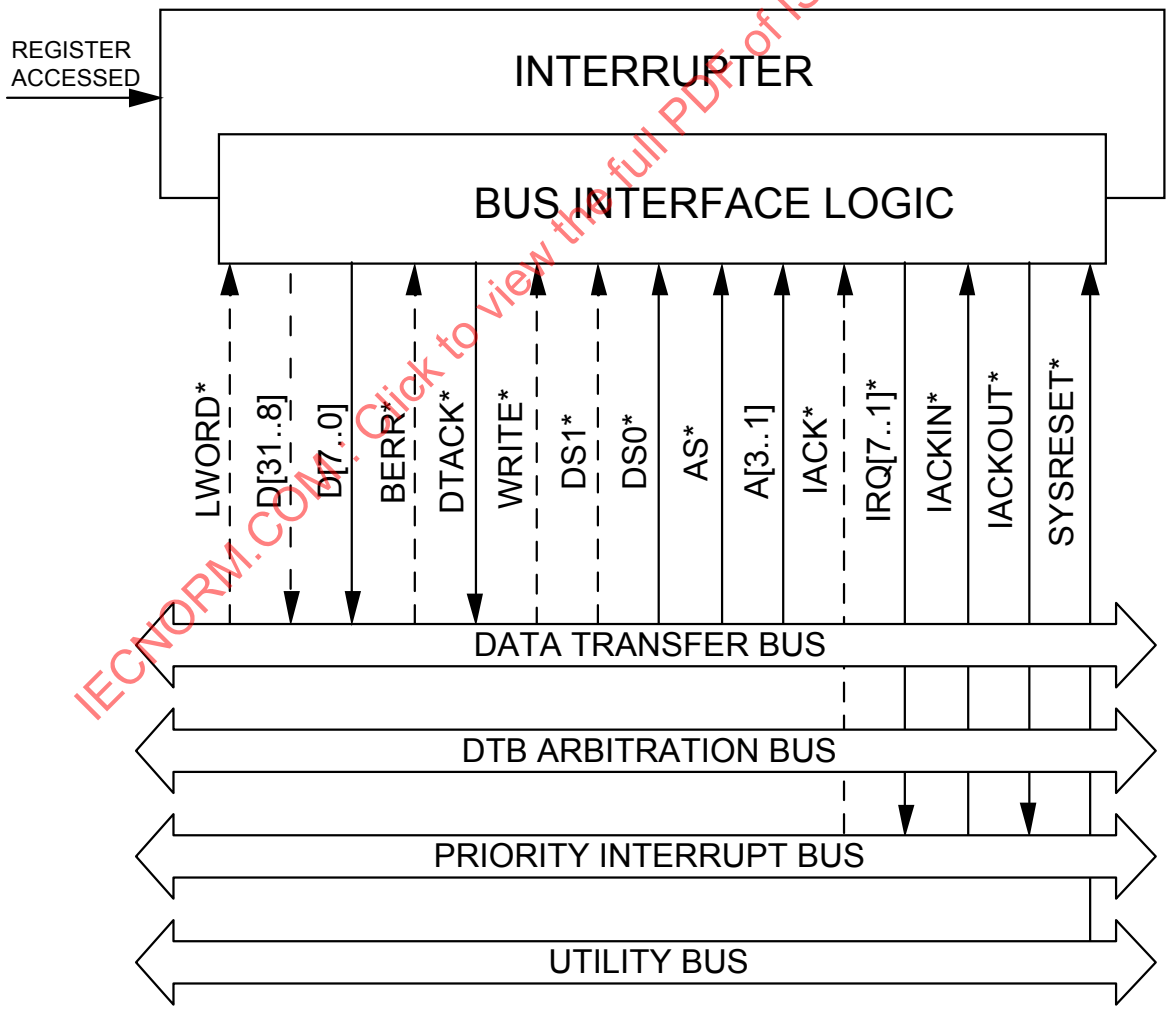
IF condition 4 is met, but either 1, 2, or 3 is not

THEN the Interrupter **MUST** pass the falling edge of IACKIN* to the next Interrupter module in the daisy-chain.

Table 36 – RULEs and PERMISSIONs specifying the use of dotted lines in Figure 52 by the various types of interrupters

Type of interrupter	Use of dotted lines
D08(O)	**MUST** drive D[7..0]. **MAY** drive D[31..8]. **MAY** monitor LWORD* or DS1*.
D16	**MUST** monitor DS1*. **MUST** drive D[15..0]. **MAY** drive D[31..16] low. **MAY** monitor LWORD*.
D32	**MUST** monitor DS1* and LWORD*. **MUST** drive D[31..0].
ALL	**MAY** monitor WRITE* and IACK*. **MAY** drive BERR*.
NOTE The mnemonics D08(O), D16, and D32 are defined in Table 39.	

A block diagram of the Interrupter is shown in Figure 52.



NOTE 1 The RULEs and PERMISSIONs for driving and monitoring the dotted lines are given in Table 36.

NOTE 2 The Register Accessed input signal is present on RORA Interrupters only.

Figure 52 – Block diagram – Interrupter

4.3.3 IACK daisy-chain driver

The IACK Daisy-Chain Driver is another module that interacts with Interrupt Handlers and Interrupters to coordinate the servicing of interrupts. It generates a falling edge on the interrupt acknowledge daisy-chain each time an Interrupt Handler initiates an interrupt acknowledge cycle.

A block diagram of the IACK Daisy-Chain Driver is given in Figure 53.

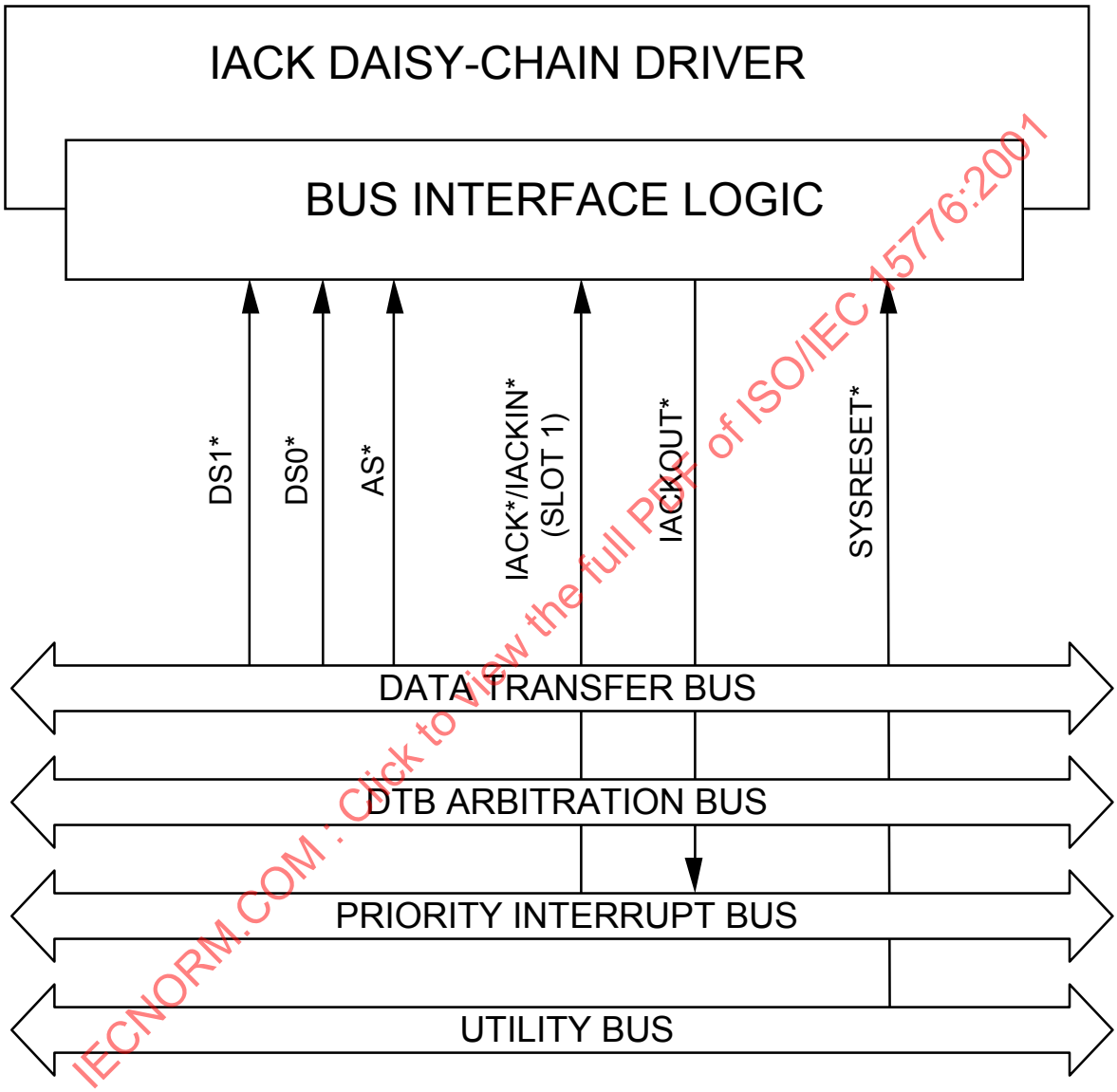


Figure 53 – Block diagram – IACK daisy-chain driver

4.3.4 Interrupt request handling capabilities

Interrupt Handlers can be designed to handle interrupt requests received on one to seven interrupt request lines. Table 37 shows how the IH( ) mnemonic is used to describe the interrupt handling capabilities of Interrupt Handlers.

Table 37 – Use of the IH( ) mnemonic to specify interrupt request handling capabilities

The following mnemonic	when applied to an	means that it
IH(x-y)	Interrupt Handler	can generate interrupt acknowledge cycles in response to interrupt requests on lines IRQx* through IRQy*.
IH(x)	Interrupt Handler	can generate interrupt acknowledge cycles in response to interrupt requests on line IRQx*.

4.3.5 Interrupt request generation capabilities

Interrupters can be designed to generate an interrupt request on any of the seven interrupt request lines. Table 38 shows how the I( ) mnemonic is used to describe the interrupt request generation capabilities of Interrupters.

Table 38 – Use of the I( ) mnemonic to specify interrupt request generation capabilities

The following mnemonic	when applied to an	means that it
I(x)	Interrupter	can generate interrupt requests on line IRQx*

4.3.6 Status/ID transfer capabilities

There are three Status/ID transfer capabilities: D08(O), D16 and D32. Table 39 shows how these mnemonics are used to describe the interrupt handling capabilities of Interrupt Handlers and Interrupters.

Table 39 – Mnemonics specifying Status/ID transfer capabilities

The following mnemonic	when applied to an	means that it
D08(O)	Interrupter	responds to 8-bit, 16-bit, and 32-bit interrupt acknowledge cycles by providing an 8-bit Status/ID on D[7..0].
	Interrupt Handler	generates 8-bit interrupt acknowledge cycles in response to the requests on the interrupt request line(s) and reads an 8-bit Status/ID from D[7..0].
D16	Interrupter	responds to 16-bit and 32-bit interrupt acknowledge cycles by providing a 16-bit Status/ID on D[15..0].
	Interrupt Handler	generates 16-bit interrupt acknowledge cycles in response to the requests on the interrupt request line(s) and reads a 16-bit Status/ID from D[15..0].
D32	Interrupter	responds to 32-bit interrupt acknowledge cycles by providing a 32-bit Status/ID on D[31..0].
	Interrupt Handler	generates 32-bit interrupt acknowledge cycles in response to the requests on the interrupt request line(s) and reads a 32-bit Status/ID from D[31..0].

4.3.7 Interrupt request release capabilities

Many widely used peripheral ICs generate interrupt requests. Unfortunately, there is no standard method for indicating to these ICs when it is time for them to remove their interrupt request from the bus. Three methods are used:

a) when the relevant processor senses an interrupt request from a peripheral device, it enters an interrupt service routine, and Reads a status register in the device. The peripheral device interprets this read cycle on its status register as a signal to remove its interrupt request;

b) when the relevant processor senses an interrupt request from a peripheral device, it enters an interrupt service routine, and Writes to a control register in the device. The peripheral device interprets this write cycle to its control register as a signal to remove its interrupt request;

c) when the relevant processor senses an interrupt request from a peripheral device, it reads a Status/ID from the device. The peripheral device interprets this read cycle as a signal to remove its interrupt request.

The VMEbus specification calls Interrupters that use methods a) and b) Release On Register Access (RORA) Interrupters, and those that use method c) Release On AcKnowledge (ROAK) Interrupters. Figure 54 shows how an ROAK Interrupter releases its interrupt request line when the Interrupt Handler reads its Status/ID and how an RORA Interrupter releases its interrupt request upon an access to a control or status register.

OBSERVATION 4.5

The Slave that provided the access to the Interrupter's control or status register is typically on the same board as the Interrupter, and it generates an on-board signal to the Interrupter when it has completed the register access.

RULE 4.5

An RORA Interrupter **MUST NOT** release its interrupt request line before it detects a falling edge on DSA* during the register access cycle. It **MUST** release the interrupt request line within 2 µs after the last data strobe goes high at the end of the register access cycle.

RULE 4.6

An ROAK Interrupter **MUST NOT** release its interrupt request line before it detects a falling edge on DSA* during the interrupt acknowledge cycle which acknowledges its interrupt, and it **MUST** release its interrupt request line within 500 ns after the last data strobe goes high at the end of the Status/ID read cycle.

RECOMMENDATION 4.2

RULEs 4.5 and 4.6 have been retained for compatibility with previous versions of the VMEbus specification. The recommended practice for new designs is that the Interrupt REQUEST line be released as soon as possible but in all cases less than 50 ns from data strobe high of the register access cycle (RULE 4.5) or the Status/ID read cycle (RULE 4.6).

OBSERVATION 4.51

The reason for speeding up the release time is to reduce the possibility of software race condition. 500 ns can now be a large number of CPU instructions.

RULE 4.7

Both RORA and ROAK Interrupters **MUST** provide a Status/ID during the interrupt acknowledge cycle that was initiated in response to their interrupt request.

RULE 4.8

After an Interrupt Handler initiates an interrupt acknowledge cycle, and reads the Status/ID from an RORA Interrupter, it **MUST** ignore the low level on the interrupt request line for 2 µs after its on-board signal REGISTER ACCESSED goes true.

OBSERVATION 4.6

RULE 4.8 prevents the Interrupt Handler from misinterpreting the low level on that line as a new interrupt request.

OBSERVATION 4.7

The Master that accesses the Interrupter's control or status register is typically on the same board as the Interrupt Handler, and it generates an on-board signal to the Interrupt Handler when it has completed the register access.

PERMISSION 4.2

IF a procedure is established to allow the Master to signal the Interrupt Handler that an access to the Interrupter's control or status registers has taken place,

THEN the Master and Interrupt Handler MAY reside on different boards.

Table 40 shows how the RORA and ROAK mnemonics are used to describe Interrupters.

Table 40 – Mnemonics specifying interrupt request release capabilities

The following mnemonic	when applied to an	means that it
RORA	Interrupter	releases its interrupt request line when a Master accesses an on-board status or control register.
ROAK	Interrupter	releases its interrupt request line when its Status/ID is read during an interrupt acknowledge cycle.

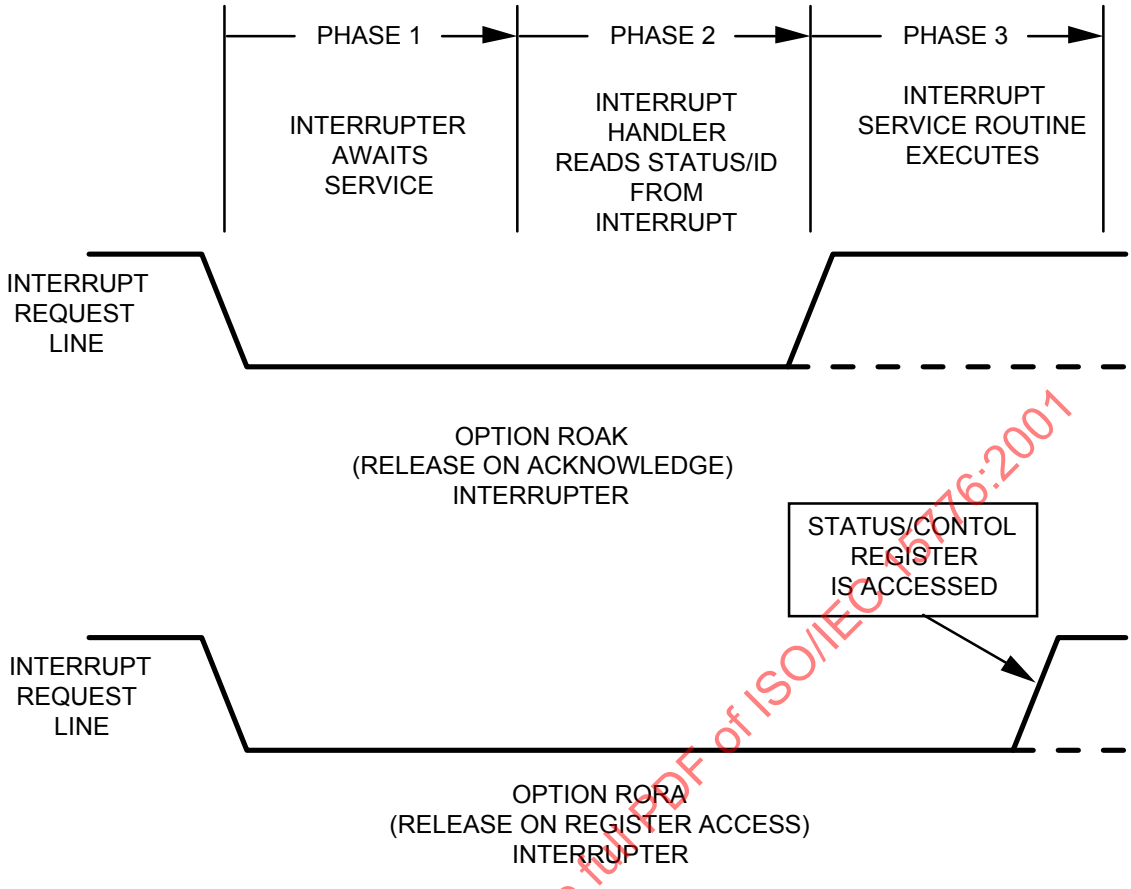


Figure 54 – Release of interrupt request lines by ROAK and RORA interrupters

4.3.8 Interaction between priority interrupt bus modules

In the following discussions, several on-board signals are defined to describe the interaction between the Interrupter and Interrupt Handler modules and other on-board logic. These signals are only intended to illustrate the information which is passed to and from the modules, rather than to define their designs.

PERMISSION 4.3

VMEbus boards MAY be designed with on-board signals that differ from those used in the following discussions.

Figure 38 shows how the IACKIN*/IACKOUT* daisy-chain is routed through a typical configuration of boards on the VMEbus. The IACK* line runs the full length of the backplane and can be driven by any Interrupt Handler that has control of the DTB. The backplane connects IACK* to the IACKIN* pin of Slot 1. The IACK Daisy-Chain Driver resides in Slot 1 and monitors the level of Slot 1's IACKIN* line.

When an Interrupt Handler drives IACK* (and Slot 1's IACKIN*) low, and then drives DSA* low, the IACK Daisy-Chain Driver generates a falling edge on its IACKOUT* pin. This pin is connected to the IACKIN* pin of Slot 2. A jumper on the board in Slot 2 routes the falling edge on the IACKIN* pin to the IACKOUT* pin, and through the backplane to the IACKIN* pin of the board in Slot 3. The Interrupter in Slot 3 does not have a pending interrupt request, so it passes on the falling edge to its IACKOUT* pin. The Interrupter in Slot 4 then detects the falling edge on its IACKIN* line and responds by placing its Status/ID on the data bus, and then driving DTACK* low.

PERMISSION 4.4

An Interrupter MAY reside on the system controller board, installed in Slot 1, along with the IACK Daisy-Chain Driver. Figure 43 shows how the two modules would be connected.

PERMISSION 4.5

More than one Interrupter MAY reside on a board. Figure 44 shows how this might be done.

OBSERVATION 4.8

In some cases, board designers might not know whether or not the board they are designing will be installed in Slot 1, or in some other slot of a VMEbus system.

RECOMMENDATION 4.1

IF a board includes both an IACK Daisy-Chain Driver and an Interrupter, and might or might not be installed in Slot 1,

THEN design it as shown in Figure 55.

PERMISSION 4.6

Several boards containing IACK Daisy-Chain Drivers MAY be installed in a VMEbus system.

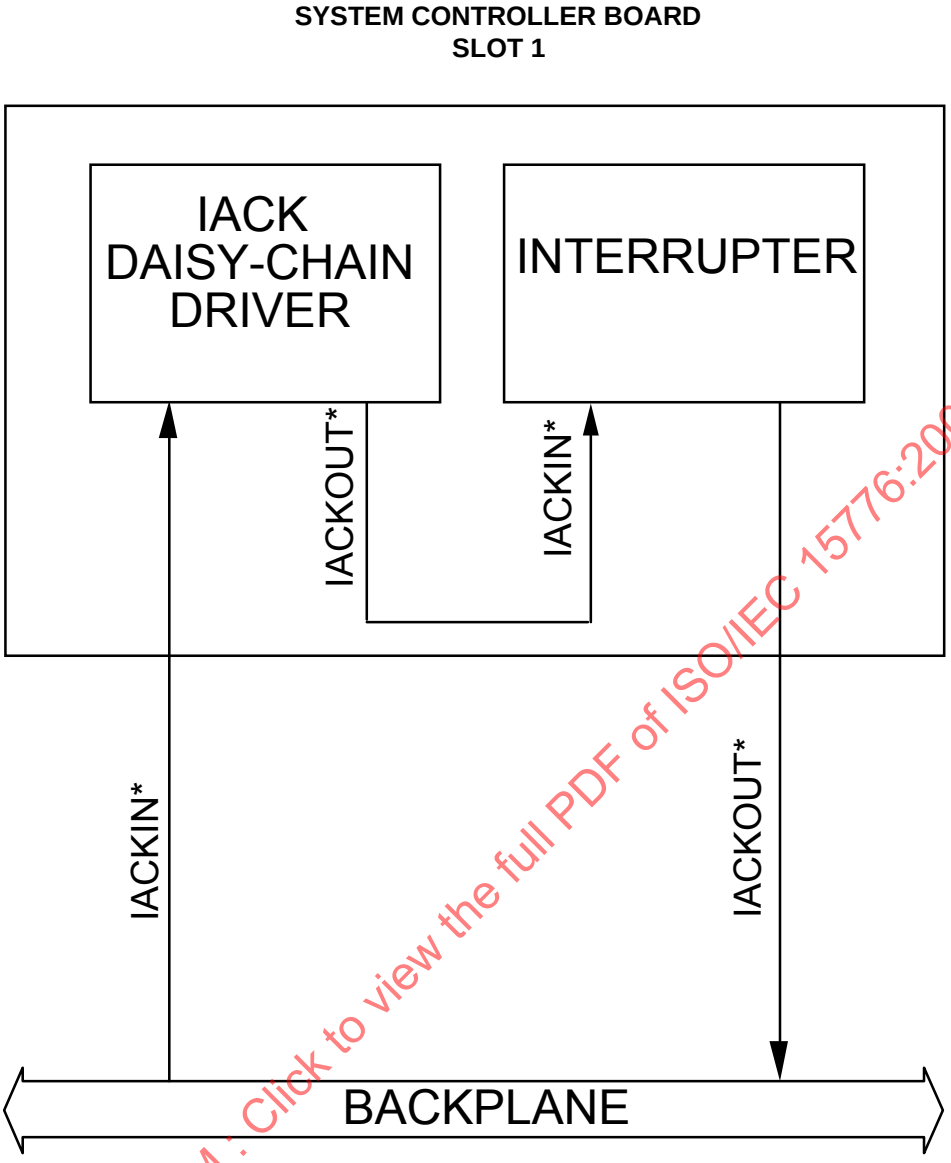


Figure 55 – IACK daisy-chain driver and interrupter on the same board

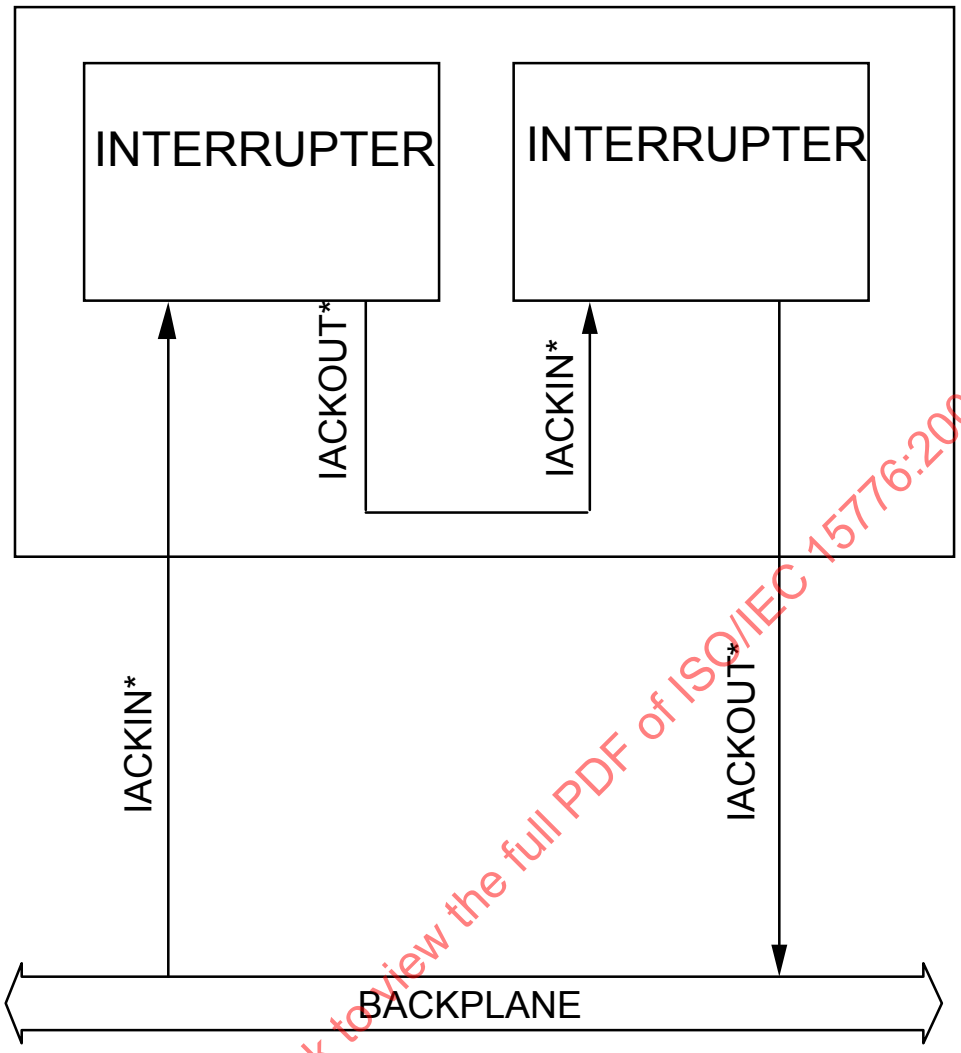


Figure 56 – Two interrupters on the same board

4.4 Typical operation

A typical interrupt sequence can be divided into three phases.

Phase 1: The interrupt request phase.

Phase 2: The interrupt acknowledge phase.

Phase 3: The interrupt servicing phase.

Figure 57 illustrates the timing relationships between the three phases.

Phase 1 starts when an Interrupter drives an interrupt request line low and ends when the Interrupt Handler gains control of the DTB. During phase 2 the Interrupt Handler uses the DTB to read the Interrupter's Status/ID. During phase 3 an interrupt servicing routine is executed. (This might or might not involve data transfers on the VMEbus.)

The protocol for the interrupt subsystem describes the module interaction required during phase 1 and phase 2. Any data transfers which take place during phase 3 will follow the Data Transfer Bus protocol described in clause 2.

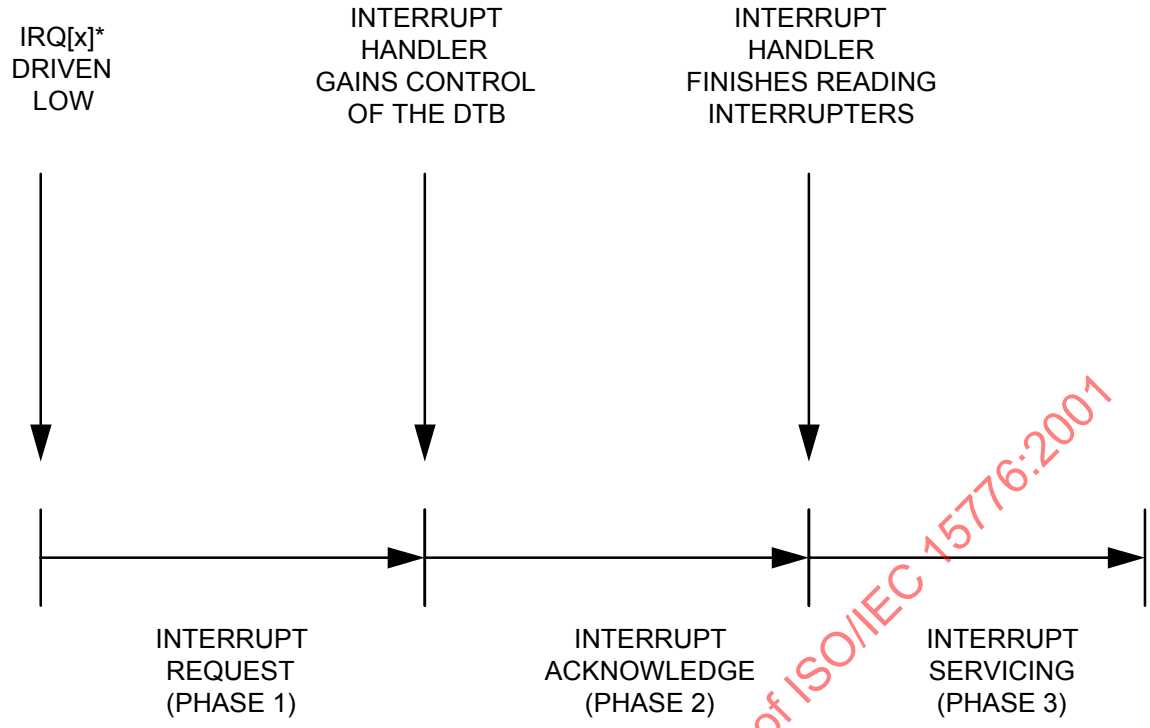


Figure 57 – The three phases of an interrupt sequence

4.4.1 Single handler interrupt operation

In single handler interrupt systems, the seven interrupt request lines are all monitored by a single Interrupt Handler. The interrupt request lines are prioritized such that IRQ7* has the highest priority and IRQ1* has the lowest priority. When the Interrupt Handler detects simultaneous requests on two interrupt request lines, it acknowledges the highest priority request first.

4.4.2 Distributed interrupt operation

Distributed interrupt systems contain from two to seven Interrupt Handlers. For the purposes of the following discussion, distributed interrupt systems will be considered in two groups:

a) distributed interrupt systems with seven Interrupt Handlers;

b) distributed interrupt systems with two to six Interrupt Handlers.

4.4.2.1 Distributed interrupt systems with seven interrupt handlers

In distributed interrupt systems with seven Interrupt Handlers, each of the interrupt request line is monitored by a separate Interrupt Handler. Each Interrupt Handler gains control of the DTB before it reads the Status/ID from Interrupters driving its interrupt request line.

OBSERVATION 4.9

There is no specified relationship between the interrupt request line that an Interrupt Handler services and the bus request line used by its on-board Requester. For example, an Interrupt Handler that services IRQ7* might have a Requester that uses BR0*, and an Interrupt Handler that services IRQ1* might have a Requester that uses BR3*. It is clear from this that there is no implied interrupt priority between lines serviced by different Interrupt Handlers.

Figure 58 illustrates a distributed interrupt system where Interrupt Handler A monitors IRQ2* and has an on-board Requester which requests the DTB on BR2*. Interrupt Handler B monitors IRQ5* and has an on-board Requester which requests the DTB on BR3*. Two Interrupters simultaneously drive IRQ2* and IRQ5* low, and the two Interrupt Handlers cause their on-board Requesters to drive BR2* and BR3* low simultaneously. In this example, priority arbitration is used and, since both bus requests go low together, the Arbiter first grants control of the DTB to Interrupt Handler B's Requester, and Interrupt Handler A waits until B has finished using the DTB.

OBSERVATION 4.10

If round-robin arbitration is used, either of the Interrupt Handlers described in Figure 58 might be granted the bus first.

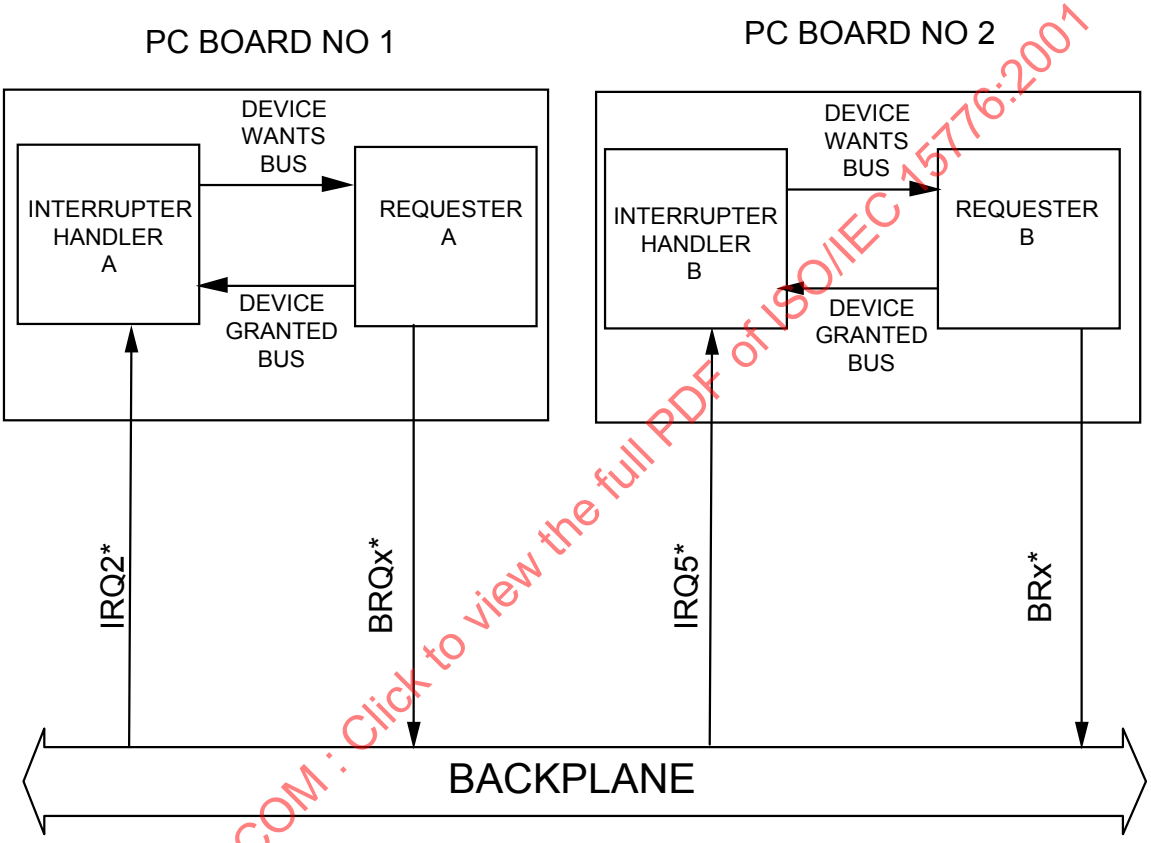


Figure 58 – Two interrupt handlers, each monitoring one interrupt request line

4.4.2.2 Distributed interrupt systems with two to six interrupt handlers

It is also possible to conFigure a distributed interrupt system in which two or more of the interrupt request lines are monitored by a single Interrupt Handler. Figure 59 illustrates a system configured with two Interrupt Handlers in which Interrupt Handler A monitors IRQ[4..1]*, and Interrupt Handler B monitors IRQ[7..5]*,. In this case, the IRQ[4..1]*, lines are prioritized; IRQ4* = highest priority for Interrupt Handler A, and the IIRQ[7..5]*, lines are prioritized; IRQ7* = highest priority for Interrupt Handler B. The DTB arbitration still determines which Interrupt Handler is granted the use of the DTB first.

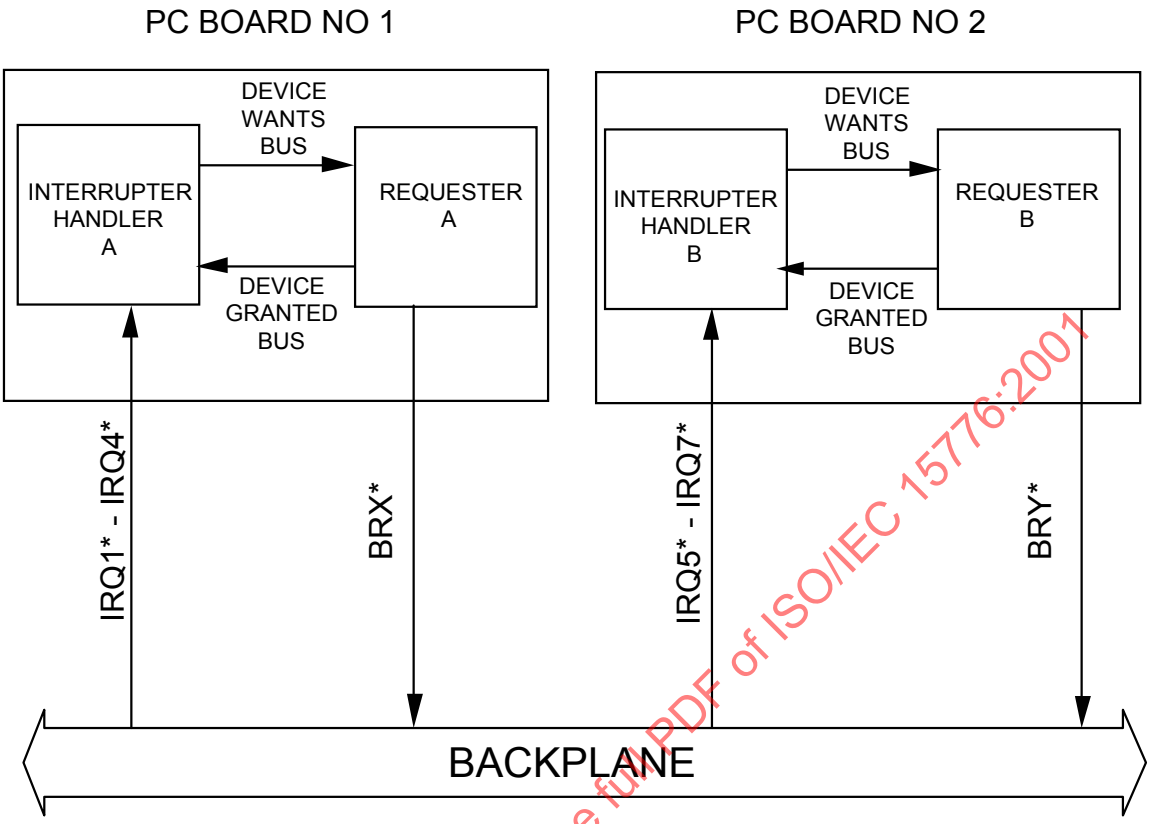


Figure 59 – Two interrupt handlers, each monitoring several interrupt request lines

4.4.3 Example: Typical single handler interrupt system operation

Figure 60 illustrates the operation of a single handler interrupt system in which one Interrupt Handler monitors and prioritizes all seven interrupt lines. At the top of the diagram, a Master is using the DTB to move data within the system at a bus request level of 2. An Interrupter in Slot 3 requests an interrupt by driving IRQ4* low. When the Interrupt Handler detects the low level on IRQ4* it sends a signal to its on-board Requester, indicating that it needs the bus. This Requester then drives BR3* low. Upon detecting the bus request, the Arbiter drives BCLR* low, indicating that a higher priority Requester is waiting for the DTB. (This example assumes a PRI Arbiter.) When Master A detects the low level on BCLR*, it stops moving data and allows its requester to relinquish control of the DTB, and release BBSY*.

OBSERVATION 4.11

The active Master is not required to relinquish the DTB within any specified time, but a prompt response to the BCLR* line allows the interrupt to be serviced quicker.

When the Arbiter detects BBSY* high, it grants the DTB to Requester B, which informs its Interrupt Handler that the DTB is available (see Figures 21 and 34). The Interrupt Handler then puts a 3-bit code on the lower three address lines, indicating that it is acknowledging the interrupt request on the IRQ4* line (see Table 41). At the same time, it drives IACK* low, indicating that it is acknowledging an interrupt, and drives AS* low. The low level on IACK* is connected, by a signal trace in the backplane, to the IACKIN* pin of Slot 1 and causes the IACK Daisy-Chain Driver to generate a falling edge down the IACKIN*/IACKOUT* daisy-chain.

When the Interrupter detects a falling edge on its incoming daisy-chain line (IACKIN*) it checks the lower three address bits to see if they match the interrupt request line which it is driving low. Since the 3-bit code matches the line on which it is making its interrupt request, when the Interrupter detects the data strobe(s) low, it places its Status/ID on the data bus and drives the DTACK* line low. When the Interrupt Handler detects the low DTACK*, it reads the Status/ID and activates the appropriate interrupt service routine.

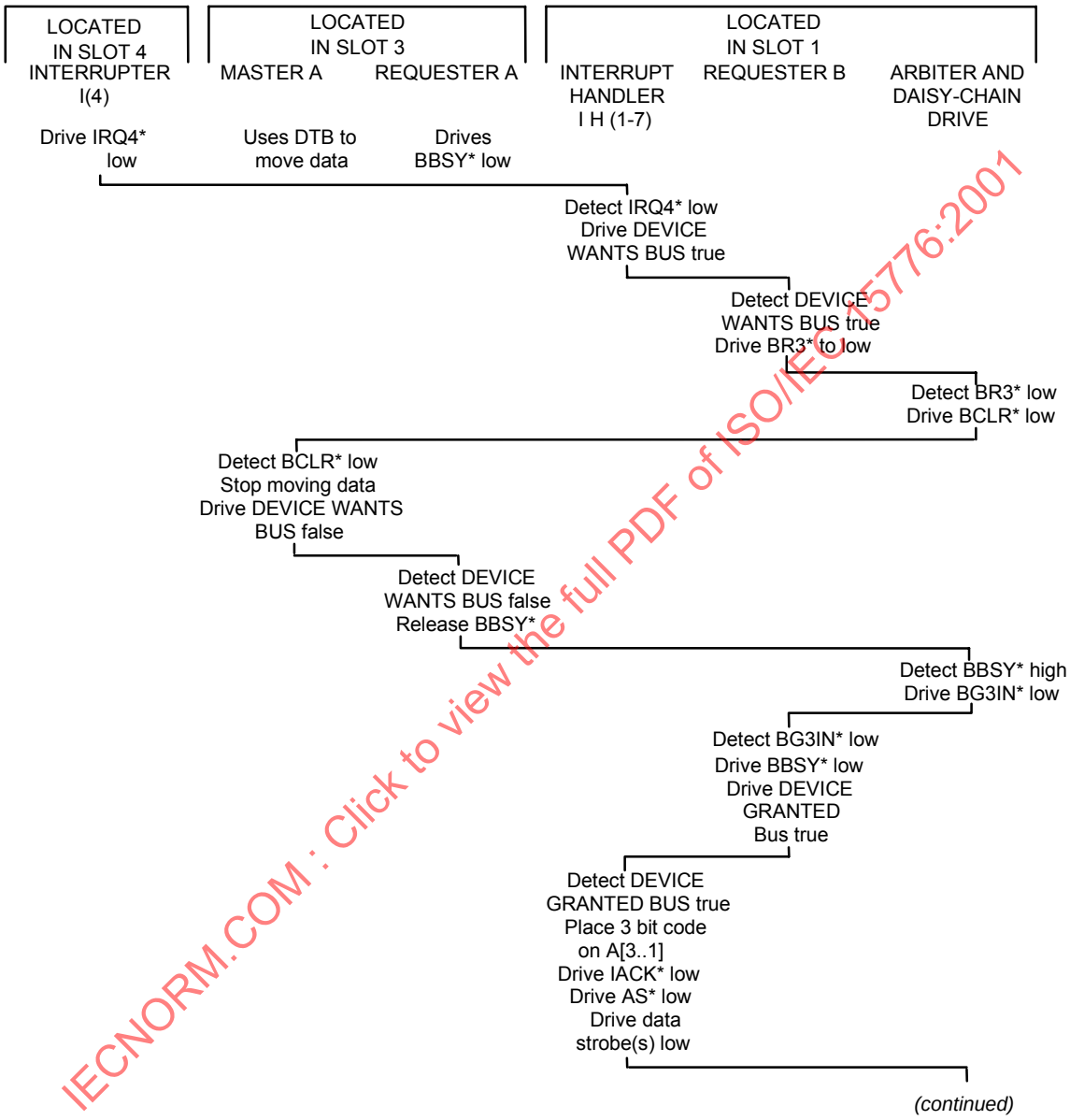


Figure 60 – Typical single handler interrupt system operation flow diagram

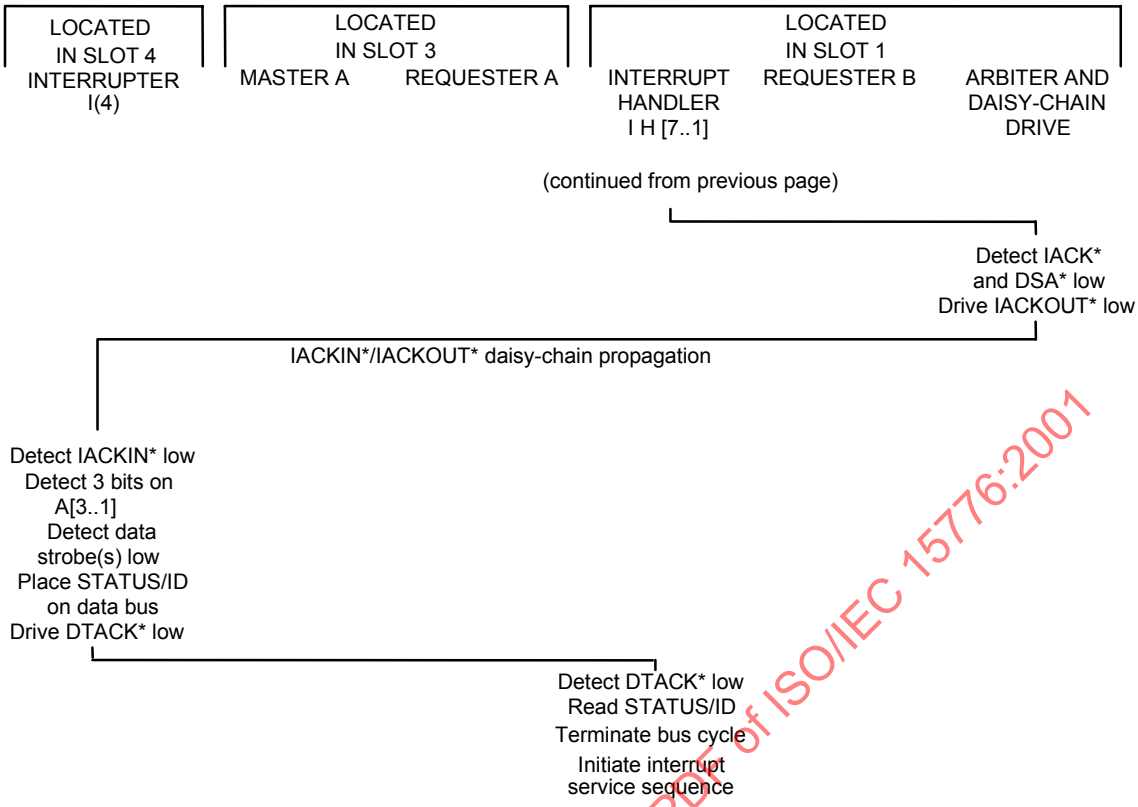


Figure 60 – Typical single handler interrupt system operation flow diagram (*continued*)

Table 41 – 3-bit interrupt acknowledge code

Interrupt line being acknowledged	Use of the address bus to broadcast the 3-bit interrupt acknowledge code		
	A3	A2	A1
IRQ1*	L	L	H
IRQ2*	L	H	L
IRQ3*	L	H	H
IRQ4*	H	L	L
IRQ5*	H	L	H
IRQ6*	H	H	L
IRQ7*	H	H	H
H = High level L = Low level			

4.4.4 Example: Prioritization of two interrupts in a distributed interrupt system

Figure 61 illustrates the operation of a distributed interrupt system with two Interrupt Handlers. Interrupt Handler A monitors IRQ[4..1]*, while Interrupt Handler B monitors IRQ[7..5]*. Interrupt Handler A treats IRQ4* as its highest priority interrupt, while Interrupt Handler B treats IRQ7* as its highest priority interrupt. At the top of the diagram, Interrupter C drives IRQ3* low, and Interrupter D drives IRQ6* low. Both Interrupt Handlers detect their respective interrupt request lines low, and both simultaneously indicate to their on-board Requester that they need the DTB. Both Requesters drive BR3* low. Upon detecting BR3* low, the DTB Arbiter drives BG3IN* low on Slot 1. This low signal is passed down the BG3IN*/BG3OUT* daisy-chain until it is detected by the Requester B in Slot 4. This Requester then signals its on-board Interrupt Handler B that the DTB is available. Interrupt Handler B then reads the Status/ID from Interrupter D.

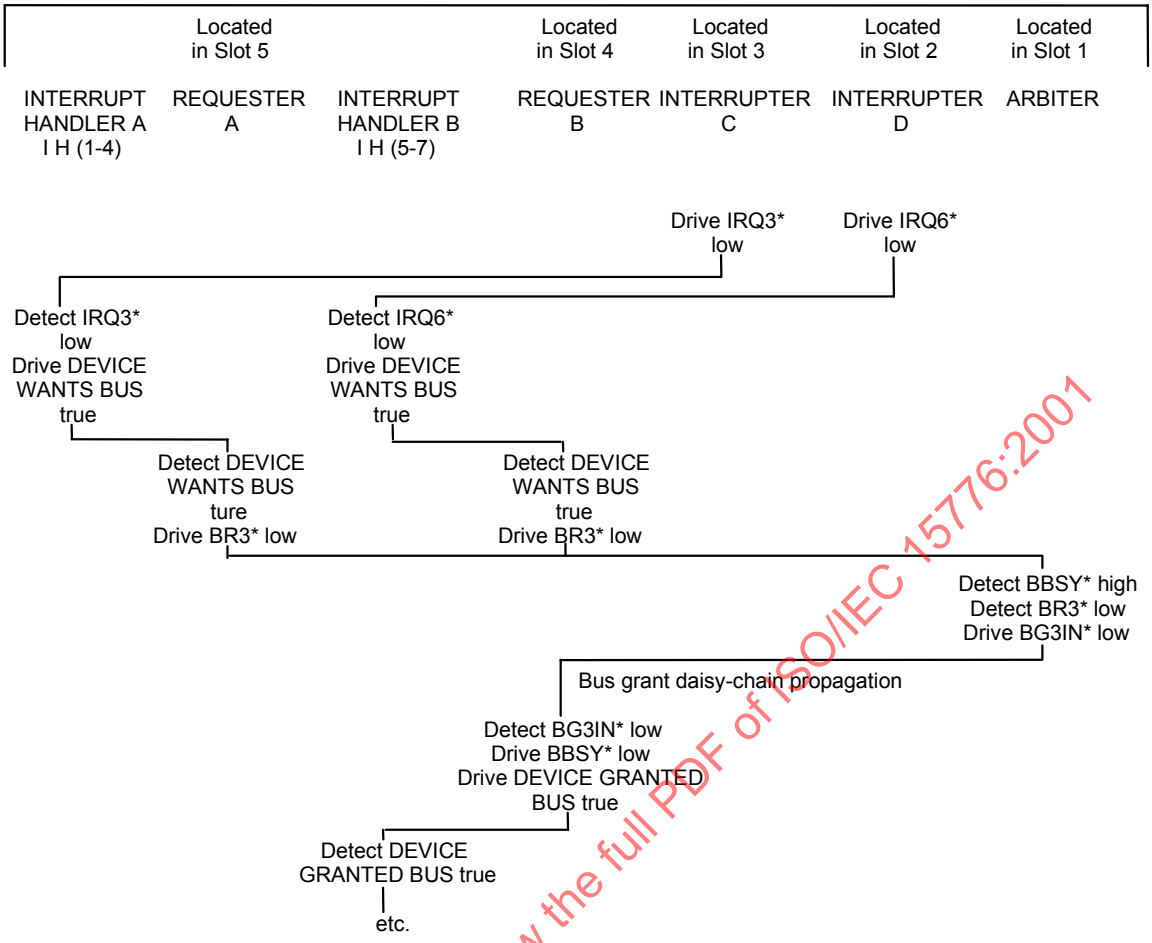


Figure 61 – Typical distributed interrupt system with two interrupt handlers, flow diagram

4.5 Race conditions

Consider the case of two interrupters A and B. Interrupter B is down the interrupt acknowledge daisy-chain from A. Interrupter B requests an interrupt. After the relevant interrupt handler is granted the bus, it acknowledges the interrupt request by driving the IACK* line low. The resulting low going edge on the interrupt acknowledge daisy-chain arrives at interrupter A at about the time it drives its own interrupt request line low. If interrupter A is improperly designed, this situation might cause it to momentarily drive its IACKOUT* line low, and then high again. This would result in a low going transient on the interrupt acknowledge daisy-chain.

RULE 4.49

Interrupters **MUST** be designed to ensure that no momentary low-going transients are generated on their IACKOUT* line.

OBSERVATION 4.50

If the interrupter is designed in such a way that it latches the state of an on-board generated interrupt request line upon the falling edge of its IACKIN* line, and if that signal is in transition when the falling edge occurs, the outputs of the latch will sometimes oscillate, or remain in the threshold region between the high and low levels for a short time. Because of this, no time limit is set for the interrupter to pass along the interrupt acknowledge. It is only prohibited from generating a low-going transient on its IACKOUT* line, which might be interpreted as an acknowledge by an interrupter further down the daisy chain.

PERMISSION 4.11

If an interrupter is about to drive an interrupt request line low between the time it receives an interrupt acknowledge intended for another interrupter and the time it would pass the interrupt acknowledge on, then it MAY treat the interrupt acknowledge as its own. In this case, the other interrupter maintains its interrupt request until another acknowledge is issued.

4.6 Priority interrupt bus timing rules and observations

The timing RULEs and OBSERVATIONs that govern the behavior of Interrupt Handlers, Interrupters, and IACK Daisy-Chain Driver during the selection of the responding Interrupter (i.e. the Interrupter that is to provide its Status/ID in response to the Interrupt Acknowledge Cycle) are described in this subclause. This timing information is given in the form of figures and tables.

The Interrupt Acknowledge Cycle begins with the selection of the responding Interrupter. This is called the Interrupter selection portion of the cycle. Once an Interrupter responds, the Interrupt Handler reads the Status/ID from it. This is called the Status/ID transfer portion of the cycle.

When the Interrupt Handler initiates an interrupt acknowledge cycle, there might be several Interrupters between it and the Interrupter being acknowledged that either:

a) do not have an interrupt pending,

b) have an interrupt pending on a different interrupt request line than the one being acknowledged.

Although these Interrupters do not respond with a Status/ID, they do participate in the Interrupt Acknowledge Cycle by passing the falling edge from their IACKIN* line on to their IACKOUT* line. For this reason, these Interrupters are called participating Interrupters.

The first Interrupter in the daisy-chain that has an interrupt pending on the interrupt request line that is being acknowledged responds with a Status/ID. For this reason it is called the responding Interrupter. All other Interrupters are called non-participating Interrupters.

Table 42 lists timing tables and timing diagrams that specify Interrupt Handler and Interrupter operation.

Table 43 lists timing tables and timing diagrams that specify IACK Daisy-Chain Driver operation.

Table 44 lists timing tables and timing diagrams that specify participating Interrupter operation.

Table 45 lists timing tables and timing diagrams that specify responding Interrupter operation.

Table 46 defines the mnemonics that are used in Tables 47 through 49.

Tables 47 through 49 specify the use of bus signal lines by the Priority Interrupt Bus functional modules.

Tables 50 through 53 specify the timing parameters for the Priority Interrupt Bus functional modules. (The reference numbers used in Tables 51 through 53 correspond to the timing parameter numbers in Table 50.)

Figures 62 through 69 are timing diagrams that specify the timing during interrupt acknowledge cycles.

Figure 70 specifies additional intercycle timing for the IACKIN*/IACKOUT* daisy-chain.

All of the RULEs and OBSERVATIONs associated with the Figures listed below also apply to Interrupt Handlers, Interrupters, and the IACK Daisy-Chain Driver.

Figures 30 through 32 in clause 2 specify the timing for the address and data strobes between data transfer cycles.

Figure 33 specifies the timing of a timed-out cycle.

Figures 34 and 21 specify the timing during mastership transfer.

In order to meet the specified timing, board designers have to take into account the worst case propagation delays of the bus drivers and receivers used on their VMEbus boards. The propagation delay of the drivers depends on their output loads. However, manufacturer specifications do not always give enough information to calculate the propagation delays under various loads. To help the VMEbus board designer, some suggestions are offered in clause 6.

The OBSERVATIONs specify the timing of signal transitions on incoming lines. These times can be relied upon as long as the backplane loading RULEs in clause 6 are not violated. The RULEs for the bus terminators in clause 6 guarantee that the timing parameters for signal lines that are released after they have been driven, are met.

Typically, for each timing RULE there is a corresponding timing OBSERVATION. However, the time that is guaranteed in the OBSERVATION might differ from the time specified by the RULE. For example, a careful inspection of the timing diagrams shows that the Interrupt Handler is required to provide 35 ns of address set-up time, but the Interrupter is only guaranteed 10 ns. This is because the address drivers are not always able to drive the backplane's signal lines completely through the low to high threshold region, until the transition propagates to the end of the backplane and is reflected back. The falling edge of the address strobe, however, typically crosses the 0,8 V threshold without waiting for a reflection. Therefore, the resulting set-up time at the Interrupter is the Interrupt Handler's set-up time less two bus propagation times.

A special notation has been used to describe the data strobe timing. The two data strobes (DS0* and DS1*) do not always make their transitions simultaneously. For purposes of these timing diagrams, DSA* represents the first data strobe to make its transition (whether that is DS0* or DS1*). DSB* represents the second data strobe to make its transition (whether that is DS1* or DS0*). The broken line shown while the data strobes are stable indicates that the first data strobe to make a falling transition might not be the first to make its rising transition, i.e., DSA* might represent DS0* on its falling edge and DS1* on its rising edge.

Table 42 – Timing diagrams defining interrupt handler and interrupter operation

Mnemonic	Type of cycle	Interrupter selection timing diagram Figure	Status/ID transfer timing diagram Figure
D08(O)	Single Byte Status/ID Read	17 & 62	66
D16	Double Byte Status/ID Read	17 & 62	67
D32	Quad Byte Status/ID Read	17 & 62	67
NOTE See Tables 62 and 63 for timing parameters.			

Table 43 – Timing diagrams defining IACK daisy-chain driver operation

Type of cycle	Interrupter selection timing diagram
Single Byte Status/ID Read	Figure 63
Double Byte Status/ID Read	Figure 63
Quad Byte Status/ID Read	Figure 63
NOTE See Tables 62 and 65 for timing values.	

Table 44 – Timing diagrams defining participating interrupter operation

Type of cycle	Interrupter selection timing diagram
Single Byte Status/ID Read	Figure 64
Double Byte Status/ID Read	Figure 64
Quad Byte Status/ID Read	Figure 64
NOTE See Tables 62 and 64 for timing values.	

Table 45 – Timing diagrams defining responding interrupter operation

Mnemonic	Type of cycle	Interrupter selection timing diagram	Status/ID transfer timing diagram
D08(O)	Single Byte Status/ID Read	Figure 65	Figure 68
D16	Double Byte Status/ID Read	Figure 65	Figure 69
D32	Quad Byte Status/ID Read	Figure 65	Figure 69
NOTE See Tables 62 and 64 for timing parameters.			

Table 46 – Definitions of mnemonics used in Tables 47, 48 and 49

Mnemonic	Description	Comments
DVBIH	Driven Valid by Interrupt Handler	**RULE 4.9** The Interrupt Handler **MUST** drive DVBIH lines to a valid level.
DLBIH	Driven Low by Interrupt Handler	**RULE 4.10** The Interrupt Handler **MUST** drive DLBIH lines to a low level.
DHBIH	Driven High by Interrupt Handler	**RULE 4.11** The Interrupt Handler **MUST** drive DHBIH lines to a high level.
dhbih?	Driven High by Interrupt Handler?	**PERMISSION 4.7** The Interrupt Handler MAY drive dhbih? lines high.
		RULE 4.12 The Interrupt Handler **MUST NOT** drive dhbih? lines low during the cycle.
dxbih?	Driven by Interrupt Handler?	**PERMISSION 4.8** The Interrupt Handler MAY drive dxbih? lines during the cycle, or it MAY leave dxbih? lines undriven. (When the line is driven it carries no valid information.)
DVBI	Driven Valid by Interrupter	**RULE 4.13** The Interrupter **MUST** drive DVBI lines a to valid level.
dhbi?	Driven by Interrupter?	**PERMISSION 4.9** The Interrupter MAY drive dhbi? lines high.
		RULE 4.14 The Interrupter **MUST NOT** drive dhbi? lines low.
dxbi?	Driven by Interrupter?	**PERMISSION 4.10** The Interrupter MAY drive dxbi? lines during the cycle, or it MAY leave the line undriven. (When the line is driven, it carries no valid information.)

Table 47 – Use of addressing lines during interrupt acknowledge cycles

Interrupt line being acknowledged	A03	A02	A01	IACK*
IRQ1*	DLBIH	DLBIH	DHBIH	DLBIH
IRQ2*	DLBIH	DHBIH	DLBIH	DLBIH
IRQ3*	DLBIH	DHBIH	DHBIH	DLBIH
IRQ4*	DHBIH	DLBIH	DLBIH	DLBIH
IRQ5*	DHBIH	DLBIH	DHBIH	DLBIH
IRQ6*	DHBIH	DHBIH	DLBIH	DLBIH
IRQ7*	DHBIH	DHBIH	DHBIH	DLBIH

Table 48 – Use of the DS1*, DS0*, LWORD* and WRITE* lines during interrupt acknowledge cycles

Mnemonic	Type of cycles	DS1*	DS0*	LWORD*	WRITE*
D08(O)	Single byte interrupt acknowledge	dhbih?	DLBIH	dhbih?	dhbih?
D16	Double byte interrupt acknowledge	DLBIH	DLBIH	dhbih?	dhbih?
D32	Quad byte interrupt acknowledge	DLBIH	DLBIH	DLBIH	dhbih?

Table 49 – Use of the data bus lines to transfer the Status/ID

Mnemonic	Type of cycles	D31-24	D23-16	D15-08	D07-00
D08(O)	Single, double, and quad byte interrupt acknowledge	dhbi?	dhbi?	dhbi?	DVBI
D16	Double and quad byte interrupt acknowledge	dhbi?	dhbi?	DVBI	DVBI
D32	Quad byte interrupt acknowledge	DVBI	DVBI	DVBI	DVBI

Table 50 – Interrupt handler, interrupter and IACK DAISY-CHAIN DRIVER timing parameters

Parameter number	Interrupt Handler See Table 51		Interrupter See Table 52		IACK Daisy- Chain Driver See Table 53	
	MIN	MAX	MIN	MAX	MIN	MAX
1	0					
2	0					
3	60					
4	35		10			
5	40		30		30	
6			0			
7			0			
9	0		0			
10	0		–10			
11	40		30			
12	35		10			
13		10		20		
14	0		0			
16	0		0			
18	0		0			
19	40		30		30	
20	0		0			
21	0		0			
23	10		0			
24A	0					
24B	0					
25		25				
26	0		0			
27	-25		0			
28	30	2T	30			
29	0		0			
30	0		0			
31	0		0			
32			10		10	
34					40	
34A			30			
35			30	0		30
36			0			
37			0			
38A			0			
38B			0			
39				40		
40			30		30	
41			0			
42					30	
43			30			

NOTE 1 All times are in nanoseconds.

NOTE 2 T = the bus time-out value.

NOTE 3 Some entries in this table are referred to in the following tables only and are not used in the timing diagrams.

Table 51 – Interrupt handler, timing RULEs and OBSERVATIONs

The numbers correspond to the timing parameters specified in Table 50.	
1	**RULE 4.15** When taking control of the VMEbus, the Interrupt Handler **MUST NOT** drive any of IACK*, A[3..1], LWORD*, WRITE*, DS0*, DS1* or AS* until after the previous Master or Interrupt Handler allows AS* to rise above the low level. **OBSERVATION 4.12** Chapter 3 describes how an Interrupt Handler's Requester is granted use of the DTB.
2	**RULE 4.16** When taking control of the DTB, the Interrupt Handler **MUST NOT** drive any of IACK*, A[3..1], LWORD*, WRITE*, DS0*, DS1*, or AS* until after it receives DEVICE GRANTED BUS true. **OBSERVATION 4.13** Chapter 3 describes how an Interrupt Handler's Requester is granted use of the DTB.
3	**RULE 4.17** When taking control of the DTB, the Interrupt Handler **MUST NOT** drive AS* low until this time after the previous Master or Interrupt Handler allows AS* to rise above the low level. **OBSERVATION 4.14** RULE 4.17 ensures that timing parameter 5 for Interrupters and Slaves is guaranteed when there is an interchange of the DTB mastership.
4	**RULE 4.18** The Interrupt Handler **MUST NOT** drive AS* low until IACK* has been low, and LWORD* and A[3..1] have been valid for this minimum time.
5	**RULE 4.19** When using the DTB for two consecutive cycles, the Interrupt Handler **MUST NOT** drive AS* low until it has been high for this minimum time.
9	**RULE 4.20** The Interrupt Handler **MUST NOT** drive DSA* low until both DTACK* and BERR* are high.
10	**RULE 4.21** The Interrupt Handler **MUST NOT** drive DSA* low before it has driven AS* low.
11	**RULE 4.22** The Interrupt Handler **MUST NOT** drive DSA* low until DS0* and DS1* have been simultaneously high for this minimum time.
12	**RULE 4.23** The Interrupt Handler **MUST NOT** drive DSA* low until WRITE* has been high for this minimum time.
13	**RULE 4.24** During double byte or quad byte Interrupt Acknowledge read cycles, the Interrupt Handler **MUST** drive DSB* low within this maximum time after it drives DSA* low. **OBSERVATION 4.15** Timing parameter 13 does not apply to single byte Interrupt Acknowledge reads.
14	**RULE 4.25** During all interrupt acknowledge cycles, the Interrupt Handler **MUST** hold the 3-bit interrupt acknowledge code valid and maintain the appropriate level on LWORD* until it detects a falling edge on DTACK* or BERR*.

Table 51 (*continued*)

16	**RULE 4.26** During all interrupt acknowledge cycles, the Interrupt Handler **MUST** maintain IACK* low until it detects a falling edge on DTACK* or BERR*.
18	**RULE 4.27** The Interrupt Handler **MUST** hold AS* low until it detects DTACK* or BERR* low.
19	**RULE 4.28** The Interrupt Handler **MUST** hold AS* low for this minimum time.
20	**RULE 4.29** Once an Interrupt Handler has driven DSA* low, it **MUST** maintain it low until it detects DTACK* or BERR* low. **OBSERVATION 4.52** During a read cycle, data may not be valid at the bus Interrupt Handler for up to 25 ns after DTACK* is detected low. (See parameter 27.) To ensure that valid data is read, the bus Interrupt Handler should maintain DSA* asserted for a least 25 ns after detecting DTACK* low.
21	**RULE 4.30** Once an Interrupt Handler has driven DSB* low, it **MUST** maintain it low until it detects DTACK* or BERR* low. See OBSERVATION 4.52
23	**RULE 4.31** Once an Interrupt Handler has driven DSA* low, it **MUST** maintain a high on the WRITE* line, until this minimum time after both data strobes are high.
24A	**RULE 4.32** **IF** the Interrupt Handler drives or releases AS* to high after its Requester releases BBSY*, **THEN** it **MUST** release IACK*, A[3..1], LWORD*, WRITE*, DS0*, and DS1* before allowing AS* to rise above the low level.
24B	**RULE 4.33** **IF** the Interrupt Handler drives or releases AS* to high before its Requester releases BBSY*, **THEN** it **MUST** release IACK*, A[3..1], LWORD*, WRITE*, DS0*, and DS1* before changing its DEVICE WANTS BUS signal from true to false.
25	**RULE 4.34** **IF** the Interrupt Handler drives or releases AS* to high after its Requester releases BBSY*, **THEN** it **MUST** release AS* within this time after allowing it to rise above the low level.
26	**OBSERVATION 4.16** Timing parameter 26 guarantees that the data bus will not be driven until the Interrupt Handler drives DSA* low.
27	**OBSERVATION 4.17** The Interrupt Handler is guaranteed that the data bus will be valid within this time after DTACK* goes low. This time does not apply to cycles where the Interrupter drives BERR* low instead of DTACK*.
28	**OBSERVATION 4.18** The Interrupt Handler is guaranteed that neither DTACK* nor BERR* will go low until this minimum time after it drives DSA* low. The Bus Timer guarantees the Interrupt Handler that if DTACK* has not gone low after its time-out period has elapsed, and within twice its time-out period, then the Bus Timer will drive BERR* low.

Table 51 (*continued*)

29	**OBSERVATION 4.19** The Interrupt Handler is guaranteed that the data bus remains valid until it drives DSA* high.
30	**OBSERVATION 4.20** This guarantees that neither DTACK* nor BERR* goes high until the Interrupt Handler drives both DS0* and DS1* high.
31	**OBSERVATION 4.21** The Interrupt Handler is guaranteed that the data bus has been released by the time DTACK* and BERR* are high.

Table 52 – Interrupter, timing RULEs and OBSERVATIONs

The numbers correspond to the timing parameters specified in Table 50.	
4	**OBSERVATION 4.22** Interrupters are guaranteed that IACK*, LWORD*, and A[3..1] have been valid for this minimum time when they detect a falling edge on AS*.
5	**OBSERVATION 4.23** All Interrupters are guaranteed this minimum high time on AS* between DTB cycles.
6	**OBSERVATION 4.24** The responding Interrupter is guaranteed that none of D[31..0] will be driven by any other module until the responding Interrupter releases DTACK* and BERR* to high.
7	**OBSERVATION 4.25** The responding Interrupter is guaranteed that the data bus will be released by all other modules by the time DSA* goes low.
9	**OBSERVATION 4.26** The responding Interrupter is guaranteed that neither DS0* nor DS1* will go low until DTACK* and BERR* from the previous cycle have gone high.
10	**OBSERVATION 4.53** Due to bus skew, Slaves on the DTB might detect a falling edge on DSA* before detecting the falling edge on AS*. However, Slaves are guaranteed that a falling edge on DSA* will not precede the falling edge on AS* by more than this time.
11	**OBSERVATION 4.27** Interrupters are guaranteed this minimum time during which both data strobes are simultaneously high between cycles.
12	**OBSERVATION 4.28** Interrupters are guaranteed that WRITE* has been high for this minimum time when they detect a falling edge on DSA*.
13	**OBSERVATION 4.29** **IF** both data strobes are going to be driven low, **THEN** the responding Interrupter is guaranteed that DSB* will go low within this maximum time after DSA*. And therefore: **IF** the DSB* does not go low within this maximum time, **THEN** the responding Interrupter assumes that it is to respond with a single byte Status/ID.

Table 52 (*continued*)

14	**OBSERVATION 4.30** The responding Interrupter is guaranteed that LWORD* and A[3..1] will remain valid until it drives DTACK* or BERR* low, provided it does so within the bus time-out period.
16	**OBSERVATION 4.31** The responding Interrupter is guaranteed that IACK* will remain low until it drives DTACK* or BERR* low, provided it does so within the bus time-out period.
18	**OBSERVATION 4.32** The responding Interrupter is guaranteed that AS* will remain low until it drives DTACK* or BERR* low, provided that it does so within the bus time-out period.
19	**OBSERVATION 4.33** Interrupters are guaranteed that the AS* will remain low for this minimum time.
20	**OBSERVATION 4.34** The responding Interrupter is guaranteed that once DSA* goes low, it will remain low until the Interrupter has driven DTACK* or BERR* low, provided that the Interrupter does so within the bus time-out period.
21	**OBSERVATION 4.35** The responding Interrupter is guaranteed that once DSB* goes low, it will remain low until the Interrupter has driven DTACK* or BERR* low, provided that the Interrupter does so within the bus time-out period.
23	**OBSERVATION 4.36** Interrupters are guaranteed that the WRITE* line remains high until both data strobes are high.
26	**RULE 4.35** The Interrupter **MUST NOT** drive the data bus until DSA* goes low.
27	**RULE 4.36** The responding Interrupter **MUST NOT** drive DTACK* low before it drives the data lines with a valid Status/ID. **OBSERVATION 4.37** This time does not apply to cycles where the Interrupter drives BERR* low instead of DTACK*.
28	**RULE 4.37** The responding Interrupter **MUST** wait this minimum time after DSA* goes low before driving DTACK* or BERR* low.
29	**RULE 4.38** Once the responding Interrupter has driven DTACK* low, it **MUST NOT** change D[31..0] until DSA* goes high.
30	**RULE 4.39** Once the responding Interrupter has driven DTACK* or BERR* to low, it **MUST NOT** release it until it detects both DS0* and DS1* high.
31	**RULE 4.40** The responding Interrupter **MUST** release all of D[31..0] before releasing DTACK* and BERR* to high.

Table 52 (*continued*)

32	**OBSERVATION 4.38** The responding Interrupter is guaranteed that IACK*, LWORD*, and A[3..1] have been valid for this minimum time when it detects a falling edge on DSA*. This time is derived from timing parameters 4 and 10.
33	**OBSERVATION 4.54** During data transfer cycles, Interrupters are guaranteed that either DS0*, or DS1*, or both remain low for at least this minimum time. This time is derived from timing parameter 28, where the responding Interrupter is required to wait a minimum time before driving BERR* or DTACK* low.
34A	**OBSERVATION 4.39** The Interrupter is guaranteed that AS* has been low for this minimum time, when it detects a falling edge on IACKIN*.
35	**RULE 4.41** A participating Interrupter **MUST** drive its IACKOUT* high within this maximum time after the rising edge on AS*.
36	**RULE 4.42** The Interrupter **MUST NOT** drive the data bus until its IACKIN* line goes low.
37	**RULE 4.43** **IF** a participating Interrupter drives the data bus, **THEN** it **MUST** release it before driving its IACKOUT* line low.
38A	**RULE 4.44** A participating Interrupter **MUST NOT** drive its IACKOUT* line low, until it detects its IACKIN* line low.
38B	**RULE 4.45** The responding Interrupter **MUST NOT** drive its DTACK* line low, until it detects its IACKIN* line low.
39	**OBSERVATION 4.40** This time guarantees that each Interrupter's IACKIN* line will go high within this time after the rising edge on AS*. This time is derived from timing parameter 35, where the IACK Daisy-Chain Driver and participating Interrupters are required to drive their IACKOUT* line high within a maximum time.
40	**OBSERVATION 4.41** All Interrupters are guaranteed that their IACKIN* line will stay high for this minimum time between consecutive DTB cycles.
41	**OBSERVATION 4.42** This time guarantees that A[3..1] and LWORD* remain valid until this time after the participating Interrupter drives its IACKOUT* low, provided it does so within the bus time-out period.
43	**OBSERVATION 4.43** This time guarantees that AS* remains low for this minimum time after the participating Interrupter drives its IACKOUT* low, provided it does so within the bus time-out period.

Table 53 – IACK daisy-chain driver, timing RULEs and OBSERVATIONs

The numbers correspond to the timing parameters specified in Table 50.	
	OBSERVATION 4.44 Since the backplane connects IACK* to the Slot 1 IACKIN*, these two signals are equivalent. Therefore, all RULEs and OBSERVATIONs that apply to one, are applicable to the other as well.
5	**OBSERVATION 4.45** The IACK Daisy-Chain Driver is guaranteed this minimum high time on AS* between DTB cycles.
19	**OBSERVATION 4.46** The IACK Daisy-Chain Driver is guaranteed that the AS* will remain low for this minimum time. This time is derived from timing parameters 8, 16, and 27 of the Interrupter.
32	**OBSERVATION 4.47** The IACK Daisy-Chain Driver is guaranteed that IACK* (and the Slot 1 IACKIN*) has been valid for this minimum time when it detects a falling edge on DSA*.
34	**RULE 4.46** **IF** the IACKIN* line is low when the IACK Daisy-Chain Driver detects a falling edge on DSA*, **THEN** it **MUST** drive its IACKOUT* line low, but it **MUST NOT** do so until this time after the falling edge on DSA*. **OBSERVATION 4.48** The IACK Daisy-Chain Driver does not drive its IACKOUT* line low every time DSA* goes low. It only does so when the IACK* line is low as well, indicating that an interrupt acknowledge cycle is in progress.
35	**RULE 4.47** **IF** the IACK Daisy-Chain Driver drives its IACKOUT* line low, **THEN** it **MUST** drive its IACKOUT* high within this time after the rising edge of AS*.
40	**RULE 4.48** The IACK Daisy-Chain Driver **MUST NOT** drive IACKOUT* low until it has been high for this minimum time.
42	**OBSERVATION 4.49** **IF** the IACK Daisy-Chain Driver drives its IACKOUT* line low within the bus time-out period, **THEN** this time guarantees that IACK* (and the Slot 1 IACKIN*) remains valid for this minimum time.

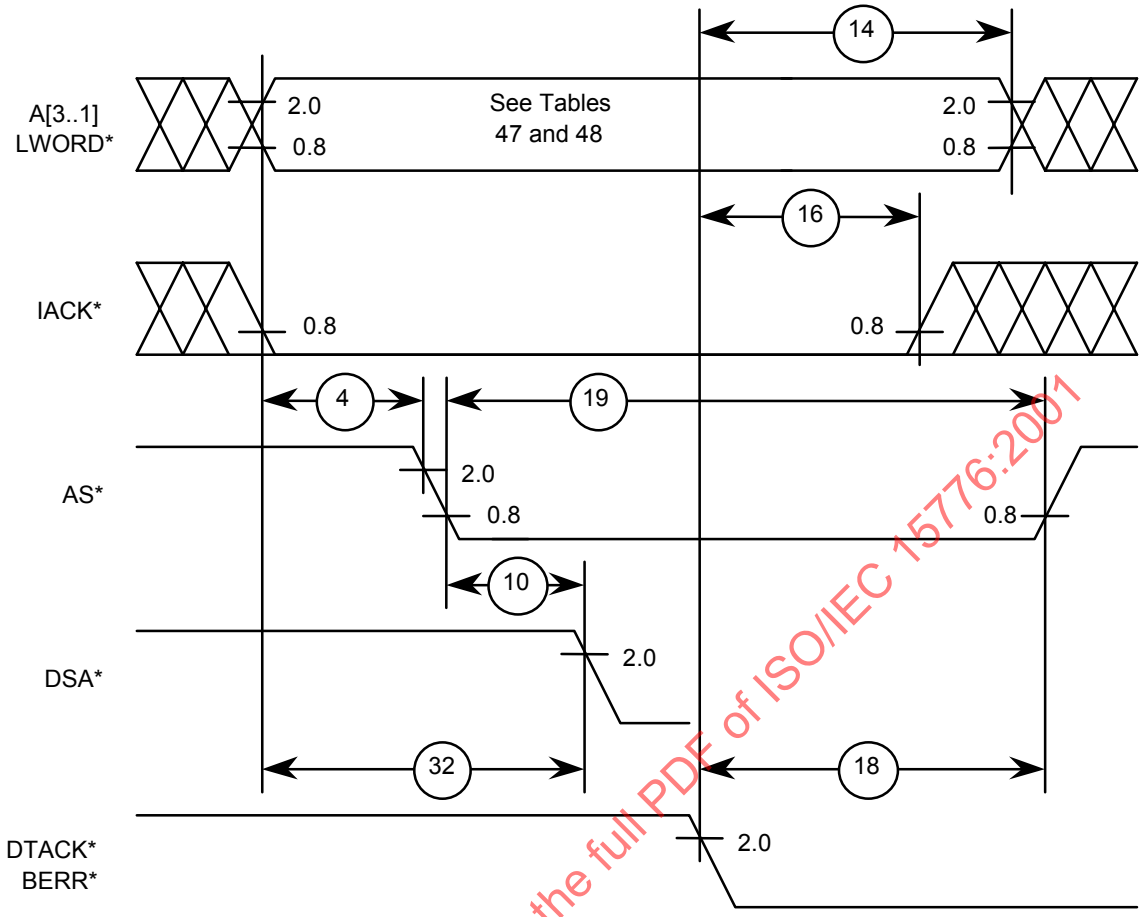


Figure 62 – Interrupt handler and interrupter – Interrupter selection timing single-byte, double-byte, and quad-byte interrupt acknowledge cycles

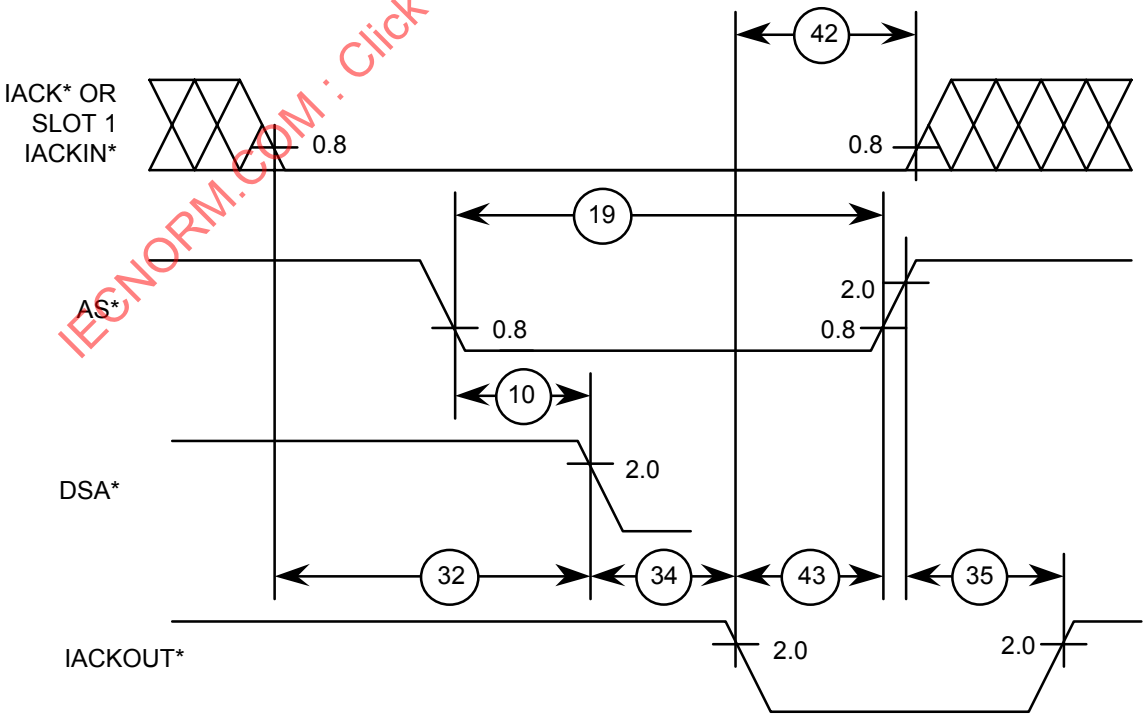


Figure 63 – IACK daisy-chain driver – Interrupter selection timing single-byte, double-byte, and quad-byte interrupt acknowledge cycles

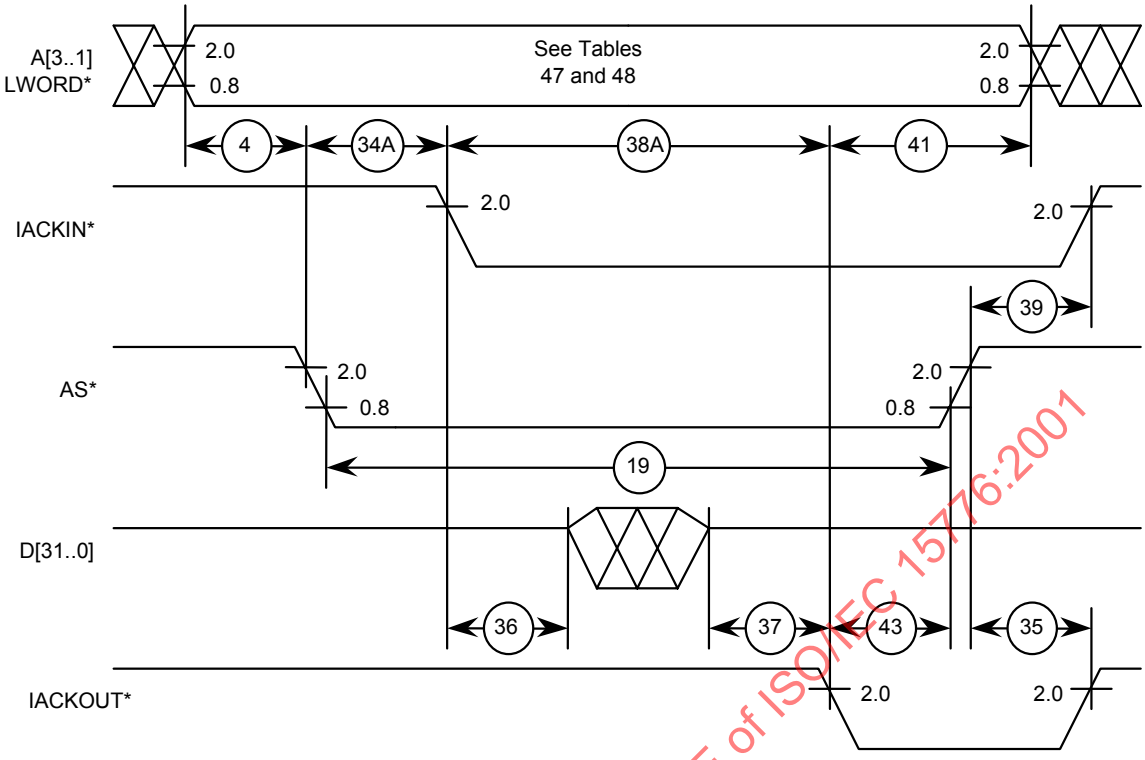


Figure 64 – Participating interrupter – Interrupter selection timing single-byte, double-byte, and quad-byte interrupt acknowledge cycles

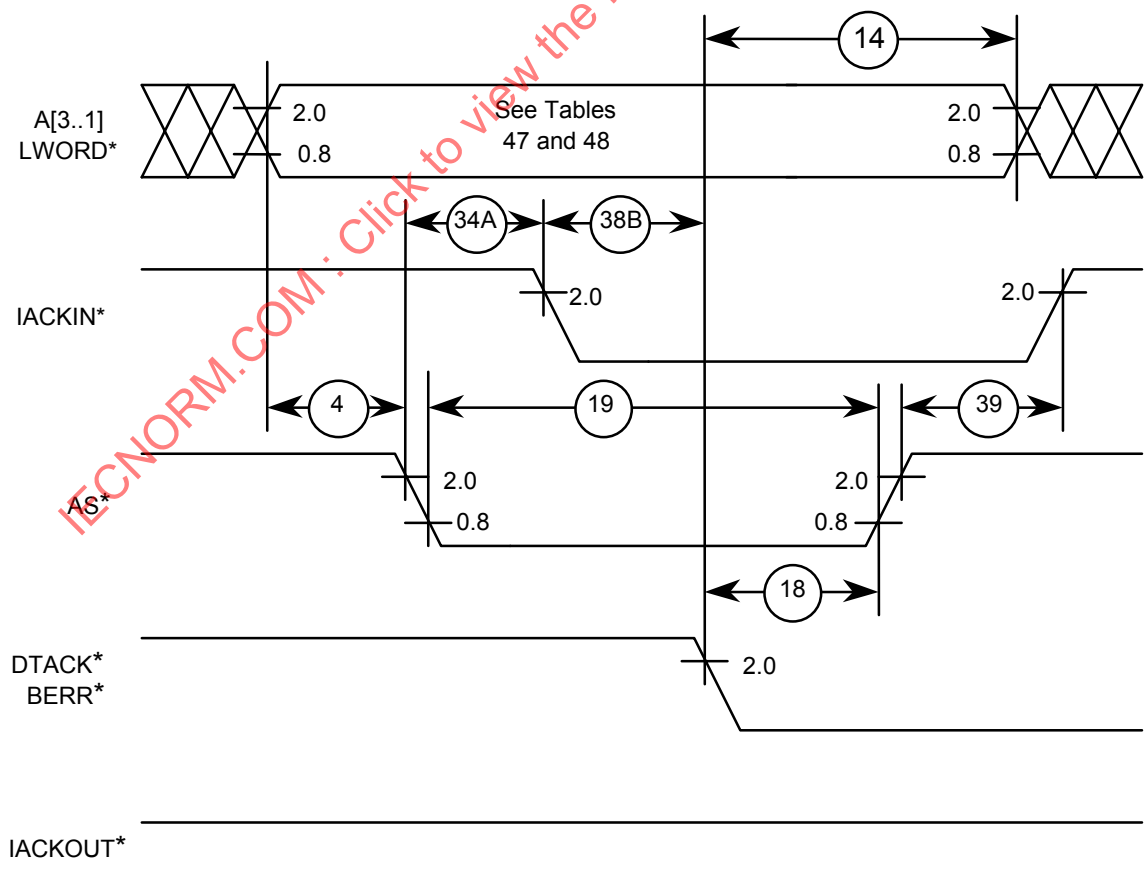


Figure 65 – Responding interrupter – Interrupter selection timing single-byte, double-byte, and quad-byte interrupt acknowledge cycles

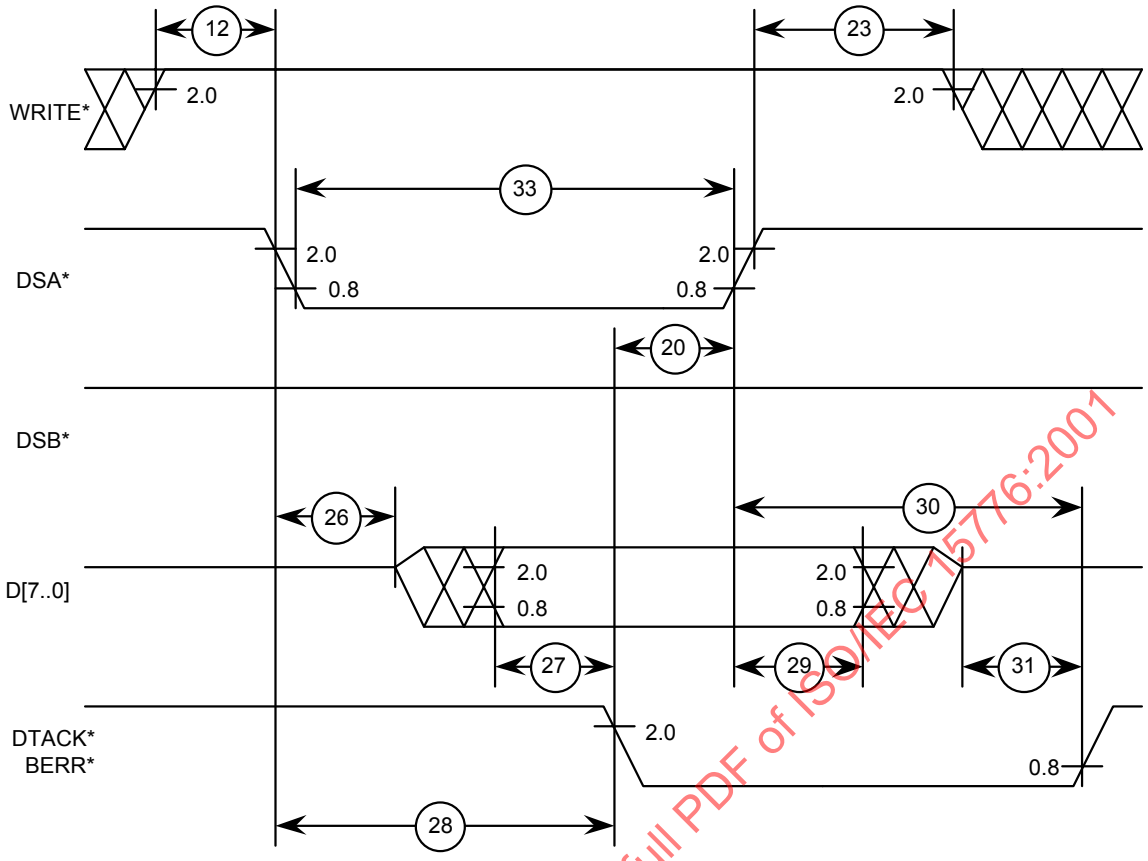


Figure 66 – Interrupt handler – Status/ID transfer timing single-byte interrupt acknowledge cycle

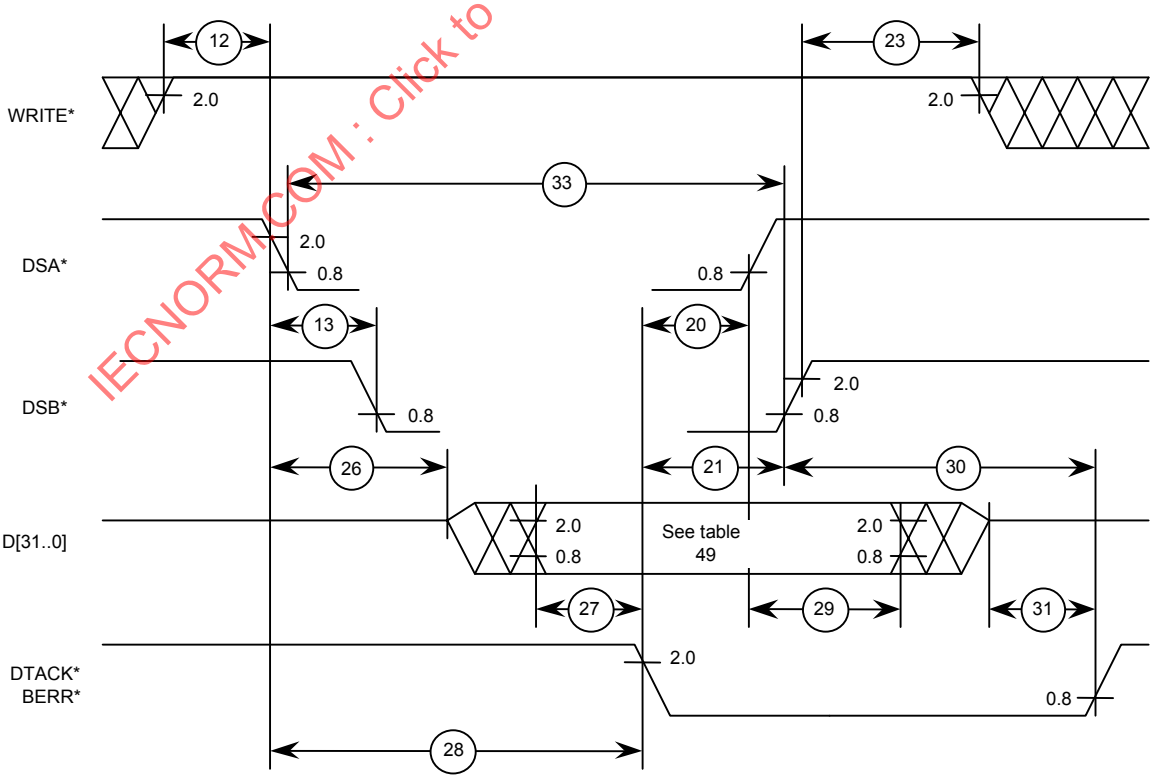


Figure 67 – Interrupt handler – Status/ID transfer timing double-byte and quad-byte interrupt acknowledge cycles

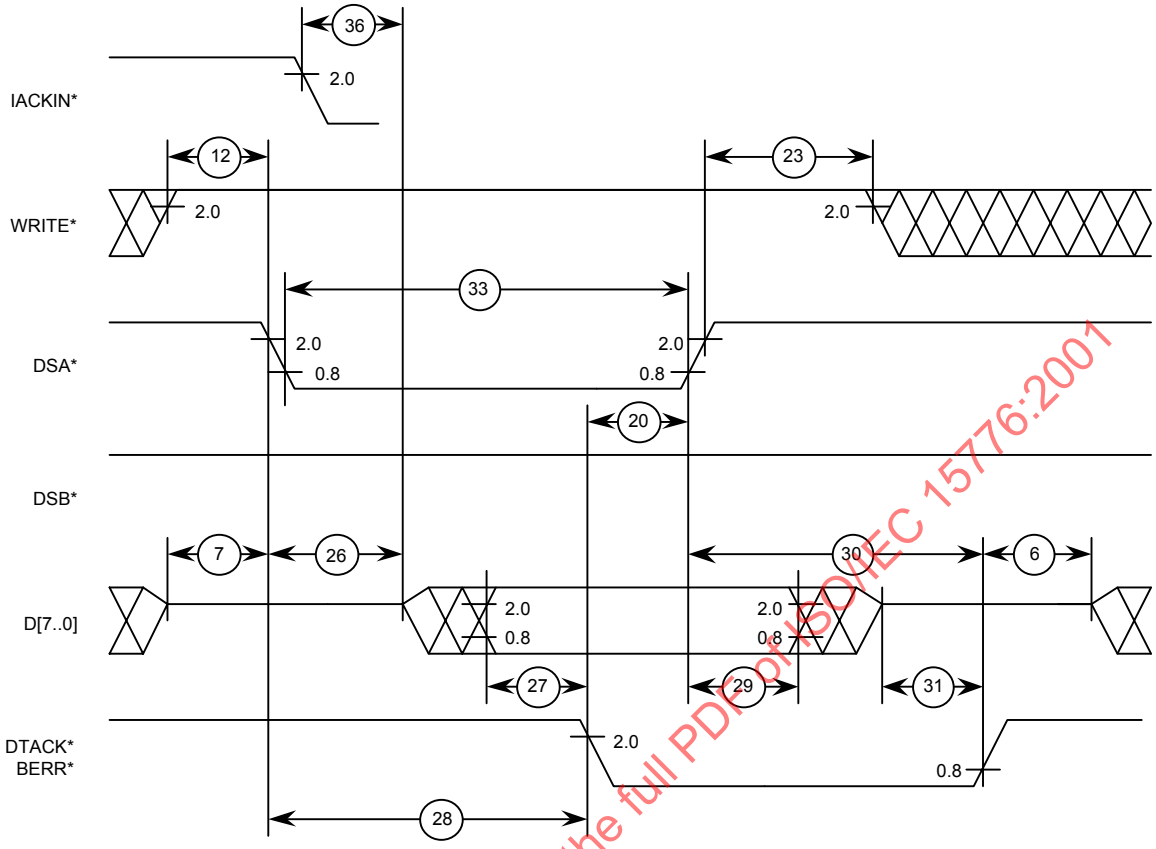


Figure 68 – Responding interrupter – Status/ID transfer timing single-byte interrupt acknowledge cycle

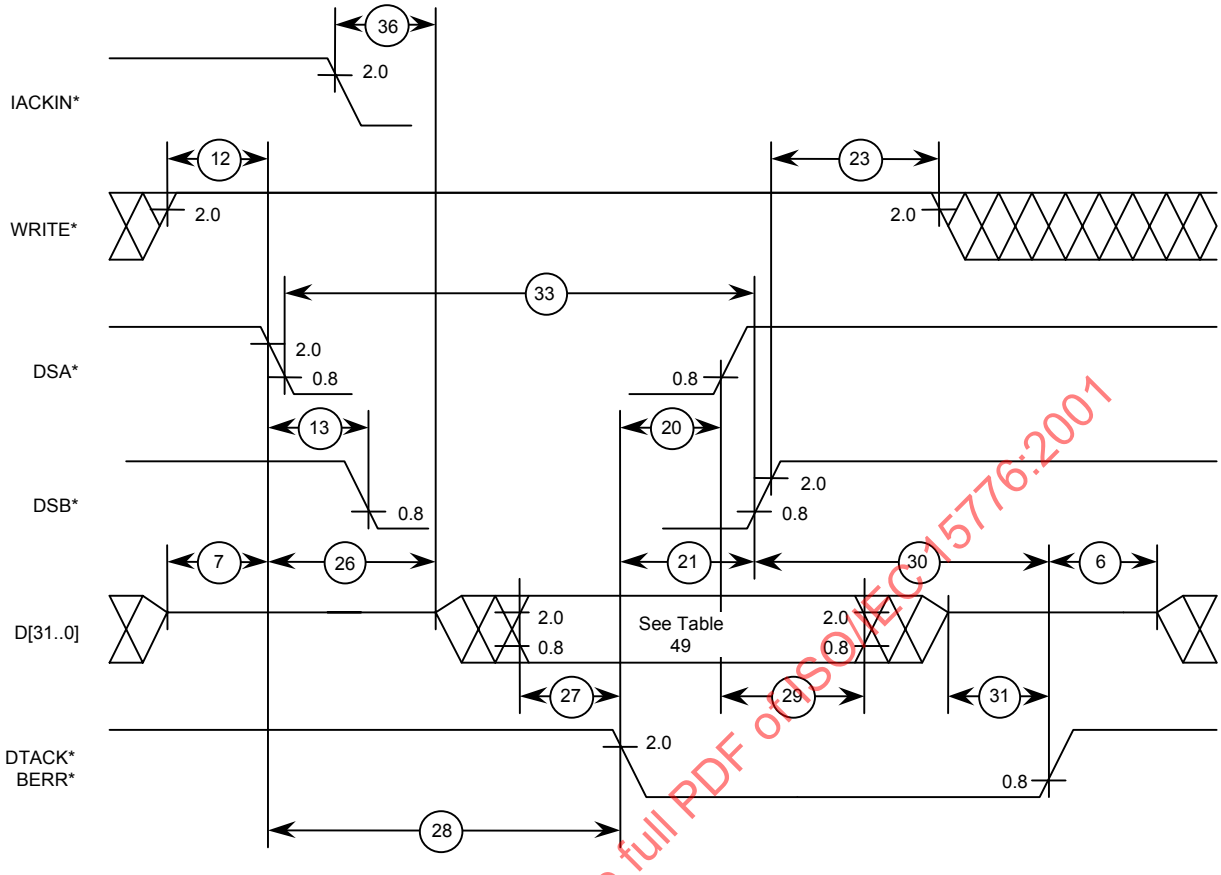


Figure 69 – Responding interrupter – Status/ID transfer timing double-byte interrupt acknowledge cycle quad-byte interrupt acknowledge cycle

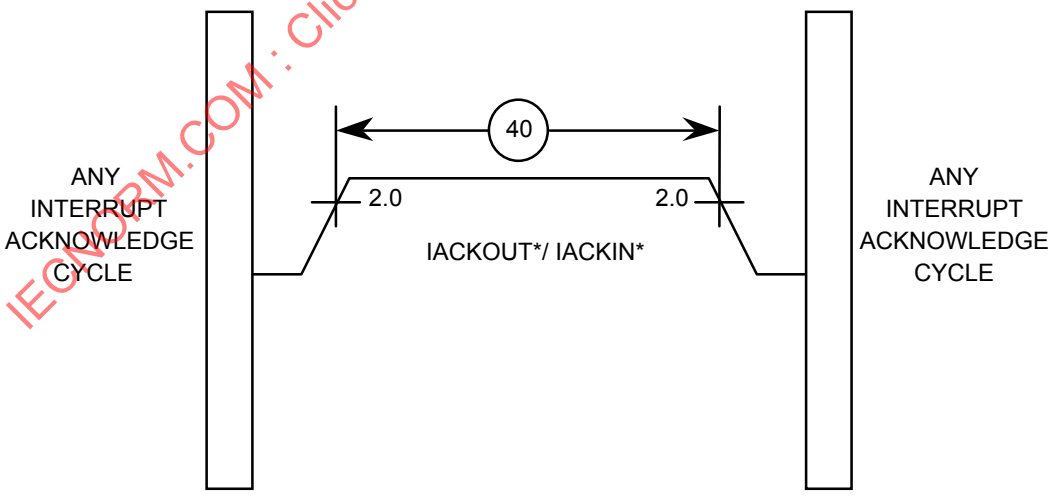


Figure 70 – IACK daisy-chain driver, responding interrupter, and participating interrupter IACK daisy-chain inter-cycle timing

5 Utility bus

5.1 Introduction

This clause identifies and defines the signal lines and modules which provide utility functions for the VMEbus. The Utility Bus supplies periodic timing, initialization and diagnostic capability for the VMEbus (see Figure 71).

5.2 Utility bus signal lines

The Utility Bus signal lines are listed below:

System Clock	(SYSCLK)
AC Fail	(ACFAIL*)
System Reset	(SYSRESET*)
System Failure	(SYSFAIL*)
Serial Bus A	(SERA)
Serial Bus B	(SERB)

5.3 Utility bus modules

5.3.1 System clock driver

The system clock is an independent, nongated, fixed frequency, 16 MHz, 50 % (nominal) duty cycle signal. The SYSCLK driver is located on the System Controller located in board slot one (see clause 1). It provides a known time base that is useful for counting off time delays. Figure 72 shows the SYSTEM CLOCK DRIVER timing diagram. For additional information see 6.5.3.

OBSERVATION 5.1

SYSCLK has no fixed phase relationships with other VMEbus timing.

5.3.2 Serial bus lines

RULE 5.11

Connector pins 'b21' and 'b22' of P1/J1 are reserved for communication via a serial bus. These pins **MUST** be bussed across the backplane and terminated in accordance with the specific implementation. Specific logical and electrical definitions relating to the serial bus are beyond the scope of this document. Serial bus implementations may include but are not limited to IEEE 1394 and AUTOBAHN technology.

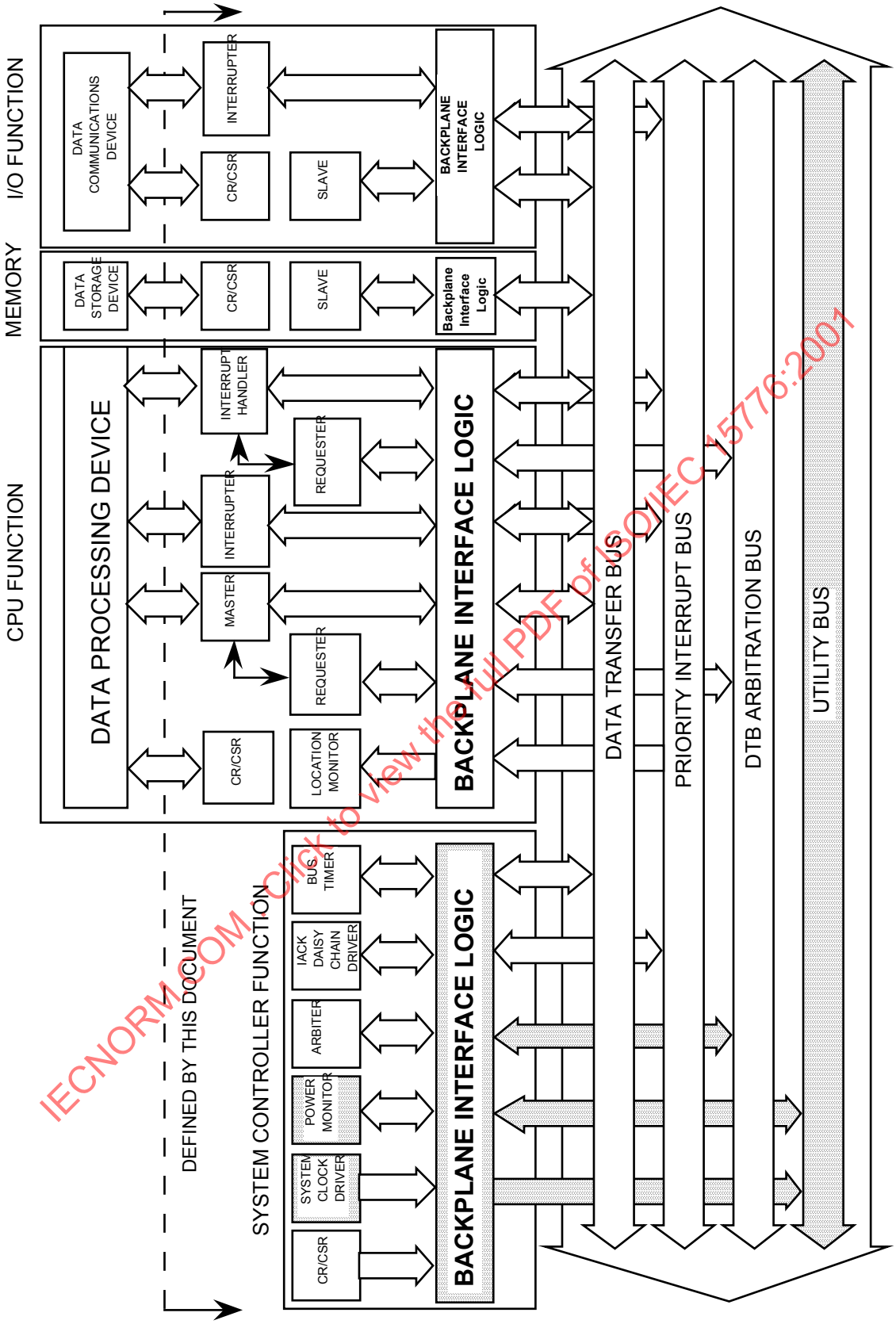
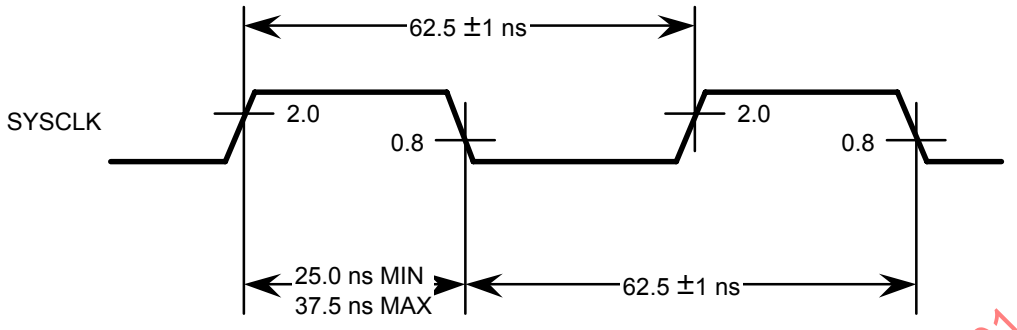


Figure 71 – Utility bus block diagram



Dimensions in nanoseconds

Figure 72 – System clock driver timing

5.3.3 Power monitor

Figure 73 is the block diagram for the Power Monitor module. This module detects power failures and signals the VMEbus system in time to effect orderly shut-down. When power is reapplied to the system the Power Monitor signals the other VMEbus modules for proper initialization.

The Power Monitor might also monitor a manually operated push button and initialize the VMEbus system whenever that button is depressed by the operator.

The ACFAIL* and SYSRESET* transitions, and the point at which the system DC voltages violate specification, have certain timing relationships. These relationships are shown in Figures 74 and 75.

PERMISSION 5.1

VMEbus systems MAY be built with or without a Power Monitor module.

Rule 5.1

Power Monitors **MUST** comply with the timing given in Figures 74 and 75.

PERMISSION 5.2

The SYSRESET* line MAY be driven low by any VMEbus board to initialize the system from a manual push-button. Where a board drives SYSRESET*, but does not drive ACFAIL*, the timing in Figure 74 and Figure 75 does not apply.

Rule 5.2

Whenever any board drives SYSRESET* low, it **MUST** hold SYSRESET* low for a minimum period of 200 ms.

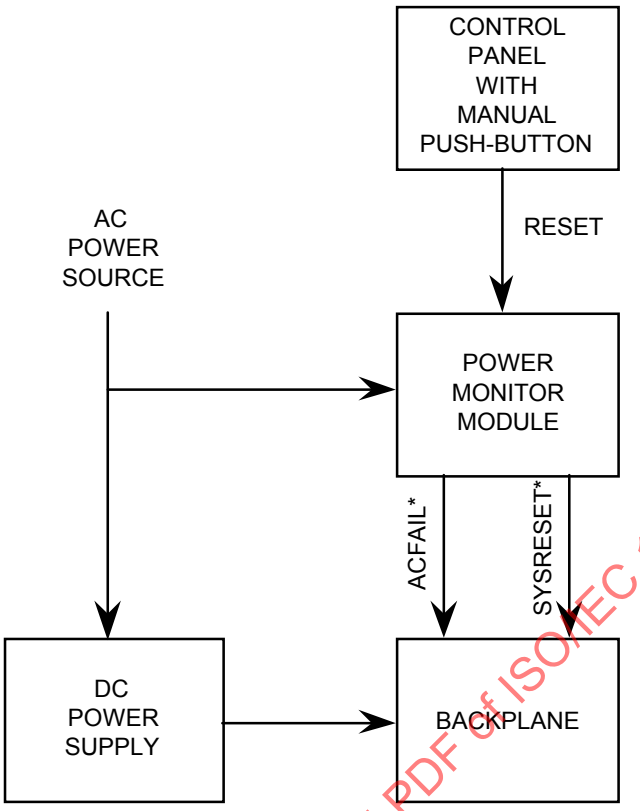


Figure 73 – Block diagram of power monitor module

5.4 System initialization and diagnostics

The VMEbus provides protocols which allow the system to be shut down and powered up in an orderly manner. Two signal lines are used in the power-down and power-up sequence: ACFAIL* and SYSRESET*. Another signal line is used in the power-up sequence: SYSFAIL*.

The following specifies the behavior of VMEbus functional modules during the power-down sequence:

RECOMMENDATION 5.1

Design Masters so that they will not request the bus for any purpose except power-fail activity, after ACFAIL* has been low for 200 μs.

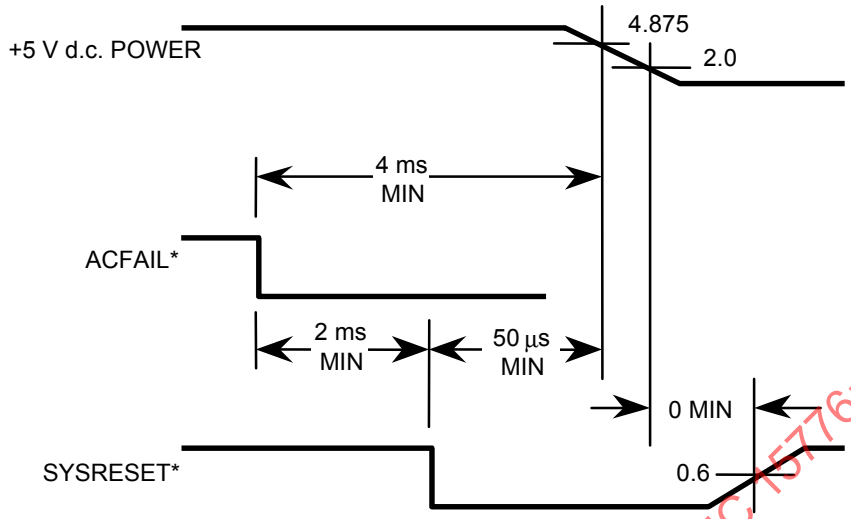


Figure 74 – Power monitor power failure timing

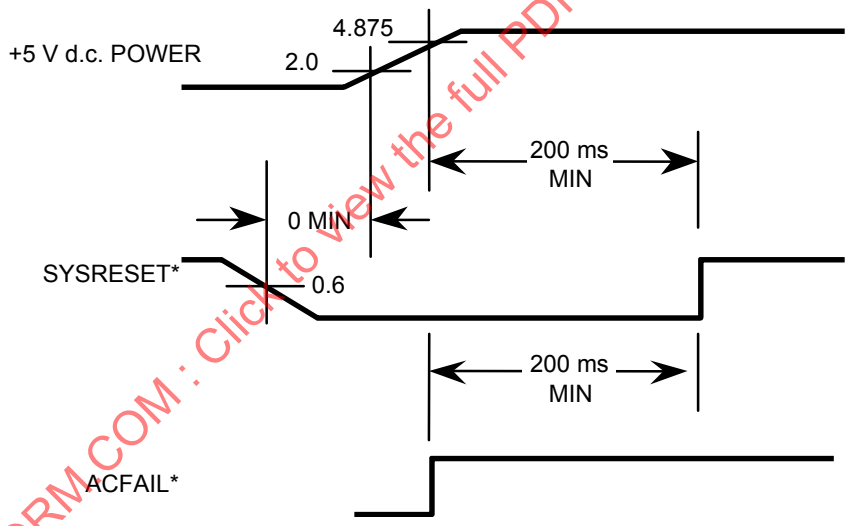


Figure 75 – Power monitor system restart timing

RECOMMENDATION 5.2

IF Masters and Interrupt Handlers have a bus request pending prior to detecting ACFAIL* low,

THEN they should limit their subsequent non-power-fail activity to 200 µs.

OBSERVATION 5.2

Bus accesses required to save and restore system data to global memory depend upon the application, and are not specified here. (The operating system has to ensure that data saved during the shutdown process is restored prior to system operation.) In the case of a multiprocessing system, this might require some interprocessor communication.

System Reset (SYSRESET*) is an open-collector line driven by the Power Monitor module, or by any board in response to a push-button switch closure.

OBSERVATION 5.3

Special circuitry is needed where push-button reset switches are used, to ensure that switch bounce does not cause the board to violate the 200 ms minimum SYSRESET* low time.

RULE 5.3

The System Clock driver **MUST** continue to provide the specified SYSCLK waveform regardless of the state of the SYSRESET* line.

PERMISSION 5.3

not used

PERMISSION 5.4

When SYSRESET* goes low, any board that requires more than 200 ms to complete its initialization may turn on its SYSRESET* driver low to maintain SYSRESET* low for the required period.

RULE 5.4

IF the +5 V d.c. power source is within its specified range when SYSRESET* goes low,

THEN functional modules **MUST** satisfy the timing RULEs given in Table 54 within the specified time after SYSRESET* goes low.

RULE 5.5

IF SYSRESET* is low when the +5 V d.c. enters its specified range,

THEN functional modules **MUST** satisfy the timing RULEs given in Table 54 and then MUST refrain from driving specified lines until SYSRESET* goes high.

RULE 5.6

After satisfying the RULEs in Table 54, functional modules **MUST NOT** change the state of their drivers until SYSRESET* goes high, unless the +5 V d.c. power source exits its specified range.

RULE 5.7

IF the +5 V d.c. power source is within the specified range when SYSRESET* goes low, and a Master or Interrupt Handler is driving AS*, DS0*, or DS1* low.

THEN it **MUST** maintain these strobes low long enough to satisfy the minimum low times given in clauses 2 and 4.

Table 54 – Module drive during power-up and power-down sequences

MODULE	MUST REFRAIN FROM DRIVING	After SYSRESET* HAS BEEN LOW FOR
Masters and Interrupt Handlers	AS*, DS0*, or DS1* from high to low	5 µs
Masters and Interrupt Handlers	IACK*, LWORD*, AS*, DS0*, DS1*,AM[5..0], A[31..1], WRITE*, or D[31..0]	20 µs
Slaves and Interrupters	D[31..0], DTACK*, or BERR*	30 µs
Interrupters	IRQ[7..1]*	30 µs
Bus Timer	BERR*	30 µs
Arbiter	BG[3..0]OUT* from high to low	5 µs
Arbiter	BG[3..0]IN* low	30 µs
Requesters	BBSY*	30 µs
Requesters	BG[3..0]OUT* from high to low	5 µs
Requesters	BG[3..0]OUT* low	30 µs

SYSFAIL* is an open collector line that is held low when the system is powered-up and remains low until system self-tests are complete (see Figure 76). The following applies:

SUGGESTION 5.1

On intelligent Master boards, include a locally accessible control register bit that is initialized to drive SYSFAIL* low when power is first applied. This permits the board's local intelligence to do a self-test and release SYSFAIL* only if the self-test passes.

SUGGESTION 5.2

Design non-intelligent boards with a globally accessible control register bit that is initialized to drive SYSFAIL* low. This allows a Master on the VMEbus to run a test on the non-intelligent board, and then write to the global control register bit, releasing the board's SYSFAIL* driver.

SUGGESTION 5.3

Where a SYSFAIL* control register bit is included on VMEbus boards, provide a status LED on the board's front panel to indicate the status of the control register bit. Then, if a system failure is indicated by the SYSFAIL* signal line, a visual inspection will help determine which board has failed.

RULE 5.8

IF SYSFAIL* control register bits are included on VMEbus boards,

THEN they **MUST** drive SYSFAIL* low within 50 ms after SYSRESET* goes low, as shown in Figure 76 and hold SYSFAIL* asserted during the assertion of SYSRESET*.

PERMISSION 5.4

A VMEbus board MAY also drive SYSFAIL* low at any time during normal operation to indicate that it has detected some kind of failure.

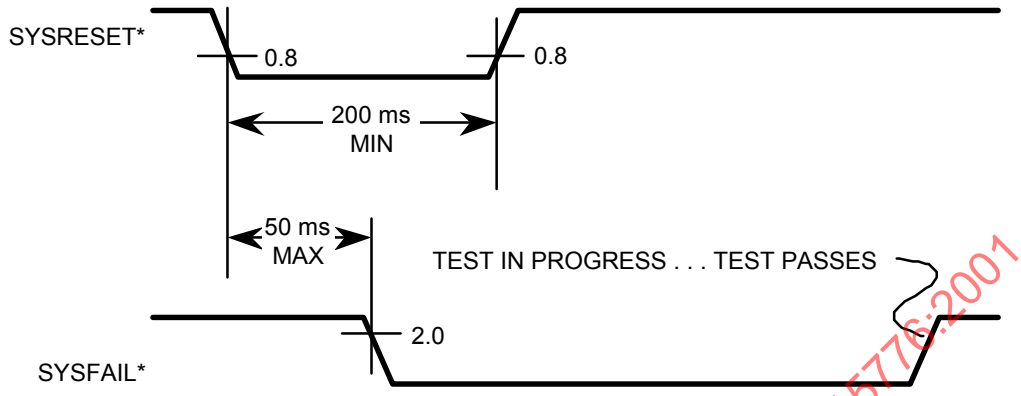


Figure 76 – SYSRESET* and SYSFAIL* timing diagram

5.5 Power and ground pins

Figure 77 gives the current rating for the VMEbus power pins at various temperatures.

OBSERVATION 5.4

Some connector pins have a slightly higher contact resistance than others when plugged into the backplane. This produces unbalanced current flow in pins which are paralleled. Suppose that 2 pins are paralleled and are carrying a total of 2 A of current. If the contact resistance on one is 1 mΩ and the other is 2 mΩ, then one pin will be carrying only 0.67 A while the other carries 1.33 A.

Rule 5.9

VMEbus connector pins **MUST** be capable of carrying the currents shown by the solid line in Figure 77.

OBSERVATION 5.5

If one or more power pins fail completely, all of the load current flows through the remaining pins. For example, if half of the pins fail, the remaining pins carry twice the normal current. Depending upon the load current, this might cause damage to these remaining good pins.

SUGGESTION 5.4

When designing a VMEbus board with a high current load, divide the board's area into zones which are each powered by a separate power grid. Don't connect these grids to each other on the VMEbus board. Instead, connect each to its own VMEbus power pin.

OBSERVATION 5.6

IF a double height VMEbus board which draws more power than its P1 connector can provide when it is plugged into a subrack which contains only a J1 backplane,

THEN its P1 power pins will overheat, and might be damaged.

SUGGESTION 5.5

Boards that support modes that drive all the address and data lines plus control signals at the same time should be designed with DIN connectors that provide 16 signal grounds on the lower outer shield.

OBSERVATION 5.8

Use of DIN connectors with signal grounds on the outer shell will improve signal integrity.

5.6 Reserved line

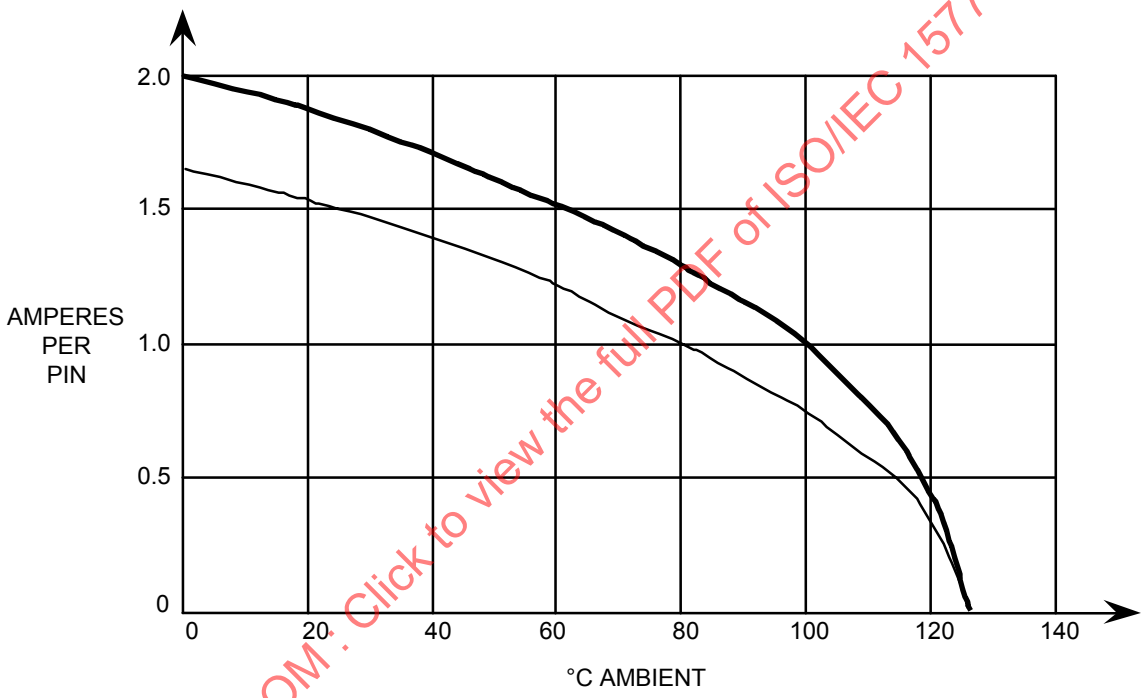
The Reserved line originally defined in VME Specification Revision C.1 is now used for the RETRY* function. Refer to clause 2.

OBSERVATION 5.7

not used.

RULE 5.10

not used.



NOTE 1 The solid bold line shows the current that can be drawn per power pin where each pin is connected to a separate power grid.

NOTE 2 The thin line shows the current that can be drawn per power pin where two or more pins are connected to a common on-board power grid.

Figure 77 – Current rating for power pins

5.7 Auto slot ID

Auto ID is an optional method of assigning the CR/CSR base address to each VMEbus board. The Auto ID Slave uses a level 2, D08(O) Interrupter along with additional board-specific hardware to obtain the CR/CSR base address. An Auto ID Master, called the Monarch, uses a level 2, D08(O) Interrupt Handler to acknowledge the Auto ID interrupt and assigns a base address to each board's CR/CSR.

Auto ID Slaves have the defined CSR register set and a CR/CSR Base Address Register located in the CR/CSR address space. Accesses to the board in the CR/CSR space are inhibited when a system reset occurs and also when the board powers up. Accesses to the board in the CR/CSR space remain inhibited until the board participates in the identification process. After power up or a system reset the Auto ID Slave drives SYSFAIL* low and inhibits CR/CSR accesses. It may also clear its CR/CSR Base Address Register to 0x00 at this time. When SYSRESET* goes high, the Auto ID Slave will prepare itself to enter into the Auto ID process. When ready to enter into the Auto ID process, the Auto ID Slave

– signals its Interrupter to generate a level 2 interrupt, and
– quits driving SYSFAIL* low.

When the Auto ID Slave's Interrupter responds to its level 2 IACK cycle, it

– enables Slave accesses in its CR/CSR space,
– presents a Status/ID byte that identifies the interrupt as an Auto ID request,
– drives DTACK* low, and
– releases IRQ2* (if a ROAK Interrupter).

The Monarch may now perform CR/CSR reads and writes to this board at A[23..19] = 0x00 in the CR/CSR space as defined by the Status/ID byte. Before responding to another IRQ2*, the Monarch reassigns the base address of this board's CR/CSR by writing a new value to this board's CR/CSR Base Address Register.

RULE 5.12

Within 5 µs after SYSRESET* goes low, each Auto ID board **MUST** inhibit accesses to its CR/CSR. It **MUST** continue to inhibit CR/CSR accesses until after it receives IACKIN* low, but before it drives DTACK* low during the interrupt acknowledge cycle in which it will provide the Status/ID byte for an Auto ID request.

RULE 5.13

Each Auto ID Slave with a CR/CSR **MUST** have a writable CR/CSR Base Address Register accessible within the board's CR/CSR address space.

RULE 5.14

An Auto ID Slave **MUST** initialize its CR/CSR Base Address Register to 0x00 before its Interrupter acknowledges the Auto ID interrupt.

RULE 5.15

A write to the CR/CSR Base Address Register **MUST** move the board's CR/CSR base address to the value written to this register before allowing DTACK* to go high at the end of the write cycle.

RULE 5.16

The Auto ID Slave **MUST** release IRQ2* before terminating the data transfer which remaps the CR/CSR or Auto ID register unless IRQ2* is being driven low for another purpose.

OBSERVATION 5.9

A ROAK Interrupter will easily guarantee RULE 5.15 in an Auto ID system. However, if the remainder of the board's functions require an RORA Interrupter, it may be desirable to use an RORA Interrupter for Auto ID requests.

PERMISSION 5.5

Either ROAK or RORA Interrupters MAY be used for Auto ID interrupt.

RECOMMENDATION 5.3

To provide interoperability between boards from different venders, the Status/ID byte for Auto ID requests should be 0xFE.

The Monarch requires a level 2 Interrupt Handler and a Master capable of initiating data transfer cycles in the CR/CSR space. After powering up or after a system reset completes, the Monarch typically waits to acknowledge interrupt requests on IRQ2* until after SYSFAIL* goes high. If the Monarch then detects IRQ2* low its Interrupt Handler initiates an Interrupt Acknowledge cycle on level 2. When the Monarch receives the Status/ID byte for an Auto ID request it

– masks IRQ2*,
– performs accesses at 0x00 in the CR/CSR space to gain information about the Auto ID Slave board,
– moves the CR/CSR to its new location, and
– unmasks IRQ2*.

Auto ID cycles do not start until the last Auto ID Slave releases the SYSFAIL* line. All Auto ID requests utilize the level 2 Interrupt Acknowledge cycle. Interrupt Acknowledge Cycles on the same level are resolved in slot-sequential order using the IACK Daisy-Chain. Therefore, if the Monarch waits until it detects SYSFAIL* high before initiating level 2 Interrupt Acknowledge cycles, then each Auto ID Slave will respond in slot-sequential order. The Monarch can use this knowledge to assign CR/CSR base addresses in sequential order to each Auto ID Slave board so that each Auto ID Slave board's CR/CSR base address relates to its relative slot position.

OBSERVATION 5.10

It is possible for a failed board to keep SYSFAIL* asserted after all Auto ID boards have quit driving SYSFAIL* low. The low SYSFAIL* will keep the Monarch from assigning CR/CSR base addresses to Auto ID Slaves.

RECOMMENDATION 5.4

Design Monarchs such that if they do not detect SYSFAIL* high within a certain amount of time, they will assume a system error has occurred and may optionally begin the Auto ID process.

OBSERVATION 5.11

Some Auto ID systems may not require knowledge of relative slot positions.

PERMISSION 5.6

In those systems which do not require knowledge of relative slot positions, the Monarch does not have to wait for SYSFAIL* to be high before responding to a low level on IRQ2*. In such a system, each Auto ID Slave MAY receive its final CR/CSR base address based on some prior agreement between the Monarch and the Auto ID Slaves.

PERMISSION 5.7

In those systems which do not require knowledge of relative slot positions, the Auto ID Slave MAY quit driving SYSFAIL* low before driving IRQ2* low to request an Auto ID.

OBSERVATION 5.12

Some systems may have a mix of Auto ID Slaves and boards with their CR/CSR base addresses at fixed or default locations in the CR/CSR space.

RECOMMENDATION 5.5

To allow for mixed Auto ID and non-Auto ID systems, the Monarch should search the system CR/CSR space for responding CR/CSR boards and either record their base addresses, or reassign their base addresses to some benign values before beginning the Auto ID process. This helps keep the Monarch from assigning the same base address to more than one CR/CSR.

Rule 5.17

The Monarch **MUST NOT** assign the same CR/CSR base address to more than one CR/CSR.

Rule 5.18

In mixed Auto ID and non-Auto ID systems, non-Auto ID boards with CR/CSRs **MUST NOT** be accessible at base address 0x00 in the CR/CSR space.

OBSERVATION 5.13

Rule 5.18 keeps non-Auto ID boards from interfering with Auto ID boards during the Auto ID sequence.

Figure 78 shows a typical Auto ID sequence for a three-slot system.

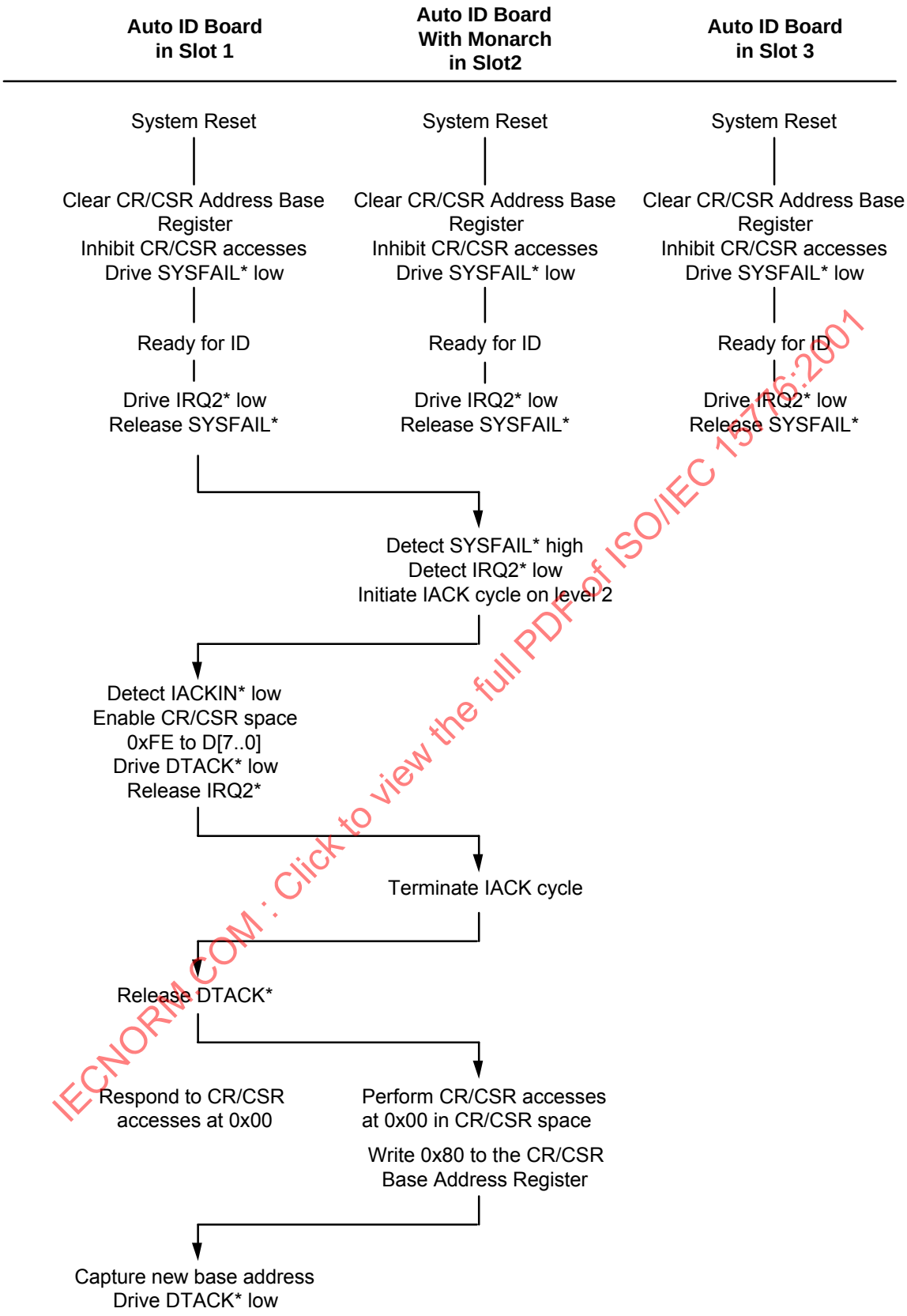


Figure 78 – CR/CSR auto ID slave initialization algorithm

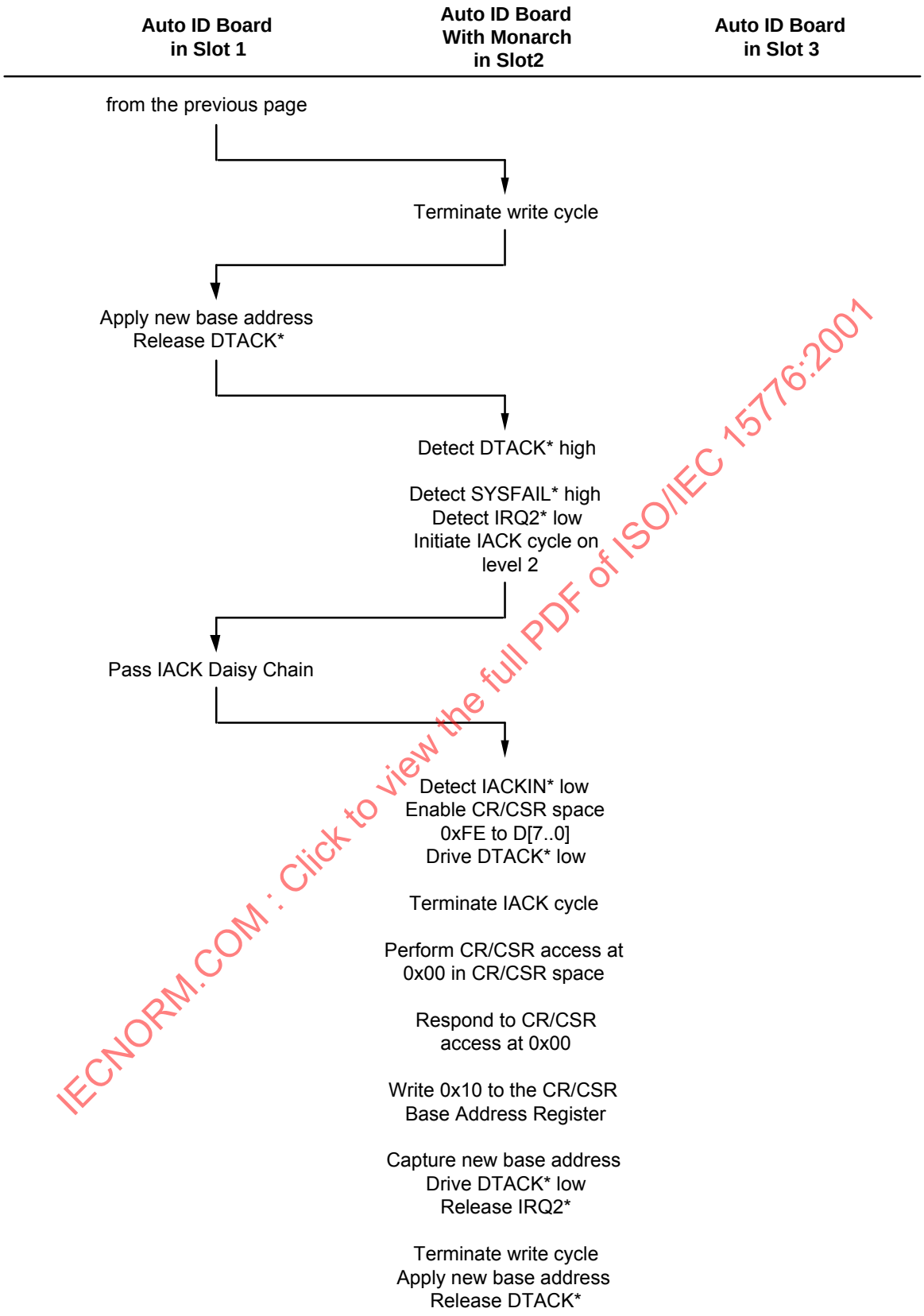


Figure 78 – CR/CSR auto ID slave initialization algorithm *(continued)*

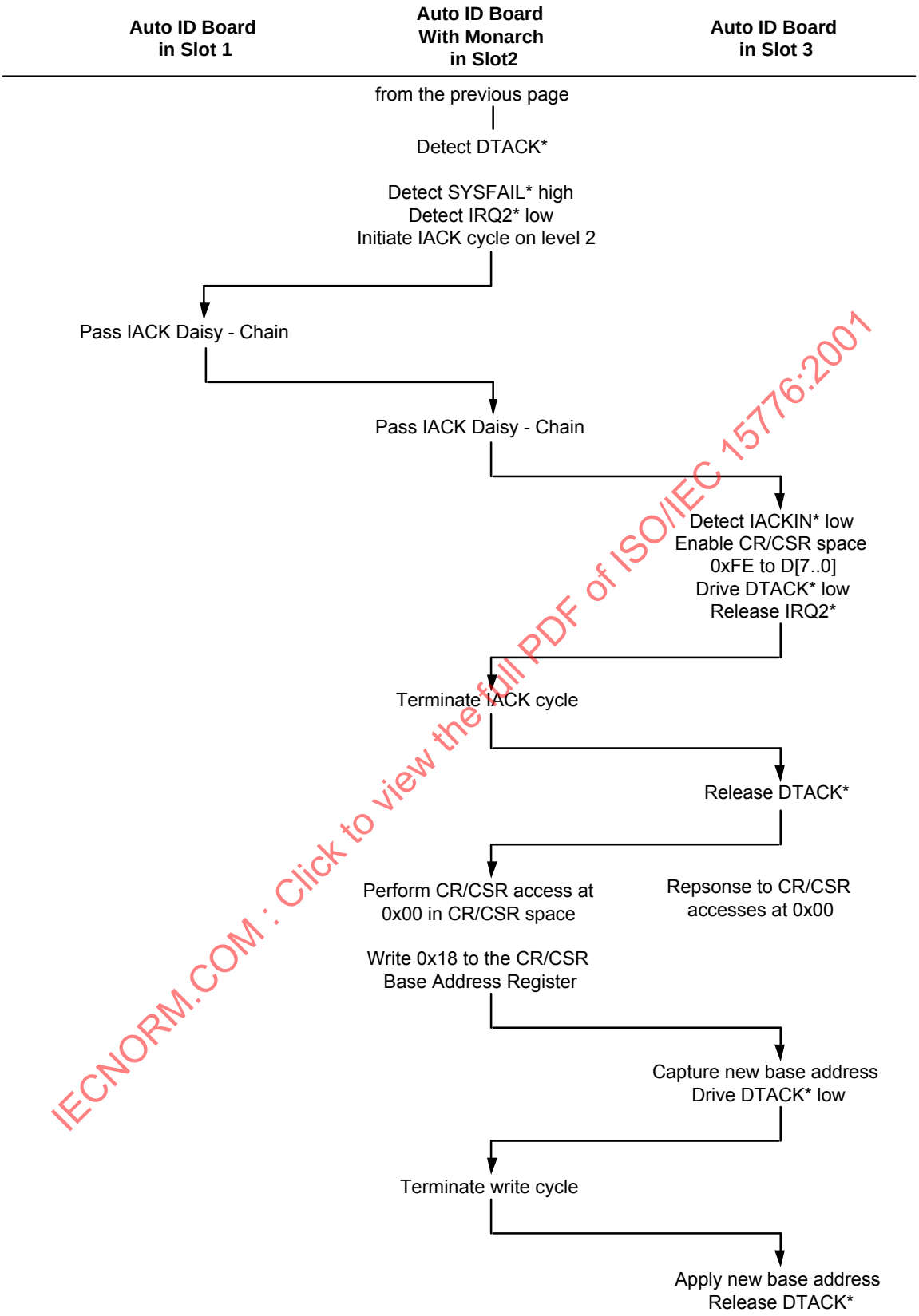


Figure 78 – CR/CSR auto ID slave initialization algorithm *(continued)*

5.8 Auto system controller

VMEbus requires that the first board in the system be configured as the System Controller. In some systems, the System Controller functionality is supplied by a separate board. In other systems, the System Controller functionality is supplied as part of the CPU boards. When included as part of a CPU board, the System Controller functions are usually enabled or inhibited by means of an on-board jumper. Configuring jumpers is a source of error in a system. The First Slot Detector (FSD) module allows a board to determine if it should enable its System Controller functions without the need for jumpers.

The typical Backplane Interface Logic on an Auto System Controller board connects a 10 kΩ resistor between BG3IN* and ground. During power-up, the FSD keeps SCON deasserted while the system voltages ramp up towards their operating potential. When the on-board logic power reaches 4.75 V, the on-board Power Monitor asserts POWER INSPEC. The FSD delays POWER INSPEC for 40 ms to allow for board-to-board Power Monitor variations, then latches the state of BG3IN*. If BG3IN* is low, the FSD asserts SCON to enable the System Controller functions. If BG3IN* is high, the FSD keeps SCON deasserted to maintain the on-board System Controller in the inhibited state.

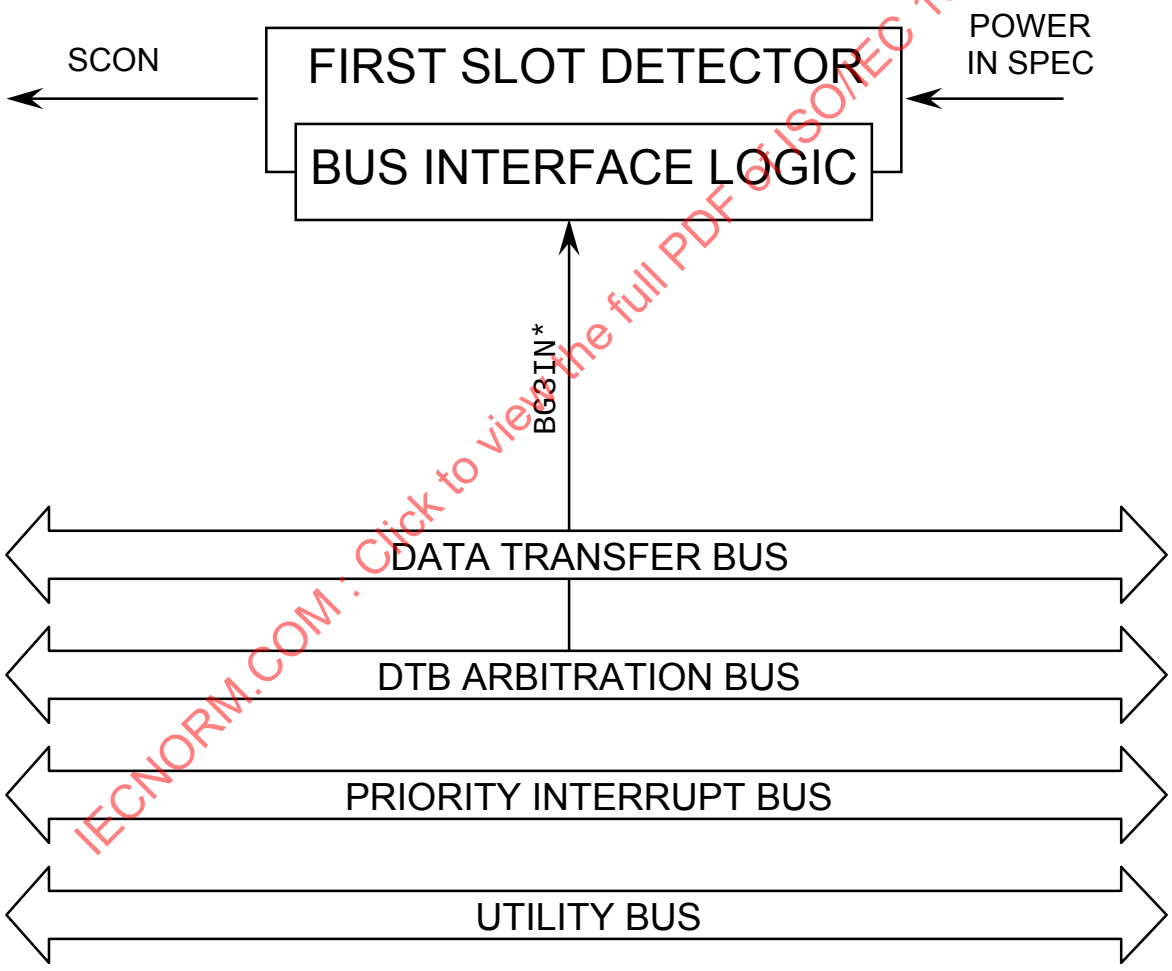


Figure 79 – FIRST SLOT DETECTOR (FSD)

RULE 5.19

Autoconfigured System Controller **MUST NOT** drive BG[3..0]IN*, or SYSCLK until the First Slot Detector (FSD) asserts SCON indicating that this board is the System Controller.

Rule 5.20

An Autoconfigured System Controller **MUST** follow the power-up sequence requirements shown in Table 54.

RULE 5.21

An Autoconfigured System Controller's Backplane Interface Logic **MUST** have a pull down on BG3IN* (see SUGGESTION 6.9).

OBSERVATION 5.14

It is necessary to use BG3IN* for the First Slot Detector because BG3IN* is the only bus grant input signal associated with a minimum function Arbiter.

RULE 5.22

Once an Autoconfigured System Controller becomes the System Controller, it **MUST** remain the System Controller until after SYSRESET* is detected low or the system is powered down. If an Autoconfigured System Controller does not become the System Controller after power-up, it **MUST** remain in the non-System Controller state until after SYSRESET* is detected low or the system is powered down.

6 Electrical specifications

6.1 Introduction

The transmission of data between VMEbus boards such as processors, memories and I/O devices takes place over one or two backplanes, depending on the design. The rules in this chapter ensure proper timing, minimal noise and minimal cross talk problems on the backplane signal lines. The design of VMEbus backplanes is governed by the following RULEs:

RULE 6.1

VMEbus backplanes **MUST NOT** have any signal conductors longer than 500 mm (19.68 in).

RULE 6.2

VMEbus backplanes **MUST NOT** have more than 21 slots.

RULE 6.3

For those lines requiring termination (see 6.7) the backplane **MUST** provide some means for terminating them at both ends of the signal line.

RULE 6.4

The backplane **MUST** provide power conductors for distribution of +5 V, +5 STDBY, +12 V, and –12 V to all of the power pins specified in 7.7.

RULE 6.5

The backplane **MUST** provide ground connections to all of the ground pins specified in Section 7.7.

PERMISSION 6.1

VMEbus signal lines are normally driven by bipolar drivers, but any technology which complies with this specification MAY be used.

6.2 Power distribution

Power in a VMEbus system is distributed on the backplane(s) as regulated direct current (DC) voltages. The available voltages are:

+5 V d.c. This is the main power source for most VMEbus systems. Most of the system circuitry, including TTL logic, MOS microprocessors, and memories, requires this voltage.

±12 V d.c. These are often used for powering RS232C drivers. They are also sometimes used for powering MOS and analog devices. In some cases –5 V d.c. bias voltage or –5.2 V d.c. ECL voltages are also derived from the –12 V d.c. source using on-board regulators. These supplies normally don't supply as much power to the VMEbus system as the +5 V d.c. source.

+5 V d.c. STDBY This is used to sustain memory, time-of-day clocks, etc., when the +5 V d.c. power is lost.

6.2.1 DC voltage specifications

Table 55 summarizes the DC voltage specifications. The listed specifications are the maximum allowed variance as measured at the connector pins of any card plugged into the backplane.

RECOMMENDATION 6.1

Design and connect backplanes so that the power supply sense point is located somewhere near the center of the backplane, and as close as possible to the point where power is introduced into the backplane.

OBSERVATION 6.1

Placing the power supply sense point near the power input point prevents boards near the power input point from receiving too high a voltage.

Table 55 – Bus voltage specification

MNEMONIC	DESCRIPTION	ALLOWED VARIATION (see OBSERVATION 6.2)	RIPPLE/NOISE BELOW 10 MHz (peak-to-peak)
+5 V	+5 V d.c.	+0.25 V / –0.125 V	50 mV
+12 V	+12 V d.c. power	+0.60 V / –0.36 V	50 mV
–12 V	–12 V d.c. power	–0.60 V / +0.36 V	50 mV
+5 V STDBY	+5 V d.c. standby	+0.25 V / –0.125 V	50 mV
GND	Ground	REFERENCE	

OBSERVATION 6.2

The non-symmetric variation given in Table 55 ensures that the DC power remains within the tolerance required by most ICs despite the typical voltage drops that occur in the power distribution network.

OBSERVATION 6.3

The power consumed by some systems fluctuates over a wide range during normal system operation. For example, dynamic memory refreshing might cause significant fluctuations if large amounts of memory are refreshed at one time. In this case the response time of the voltage distribution system becomes important.

RECOMMENDATION 6.2

Use bypass capacitors on VMEbus boards to minimize the effects of power transients.

6.2.2 Pin and socket connector electrical ratings

RULE 6.6

The 96 pin connector used by the VMEbus **MUST** provide the following:

Voltage rating:	>100 V d.c., isolation pin to pin
Contact resistance:	<50 mΩ, at rated current
Insulation resistance:	>100 MΩ, pin to pin

6.3 Electrical signal characteristics

RULE 6.7

VMEbus boards **MUST NOT** drive any backplane signal line to a higher steady-state voltage than the highest voltage on any of its +5 V power pins, or to a lower steady-state voltage than the lowest voltage on any of its GND pins.

RULE 6.8

VMEbus boards **MUST** use drivers and receivers that meet the following characteristics:

Steady-state driver	low output level	<0.6 V
Steady-state receiver	low input level	<0.8 V
Steady-state driver	high output level	>2.4 V
Steady-state receiver	high input level	>2.0 V

Figure 80 gives a simple graphic representation of these levels.

VMEbus boards drive the backplane lines with three-state, open collector, and totem-pole drivers. Subclause 6.4 specifies the drive and loading requirements for the various signal lines. Subclause 6.7 provides a summary, showing which types of drivers are used to drive each signal line.

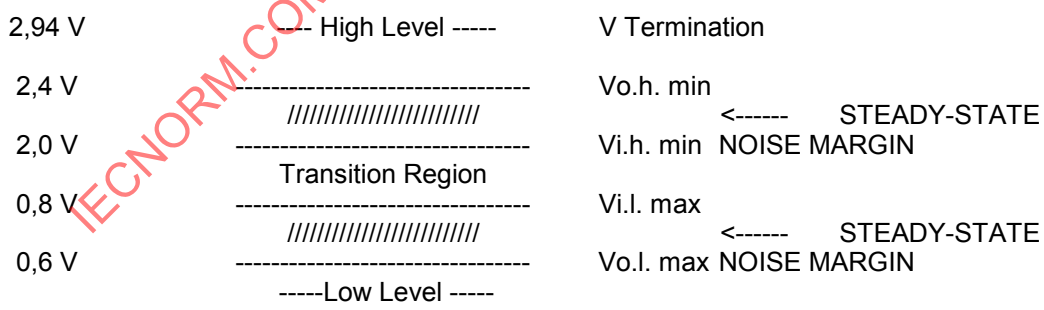


Figure 80 – VMEbus signal levels

RULE 6.9

When making voltage threshold measurements on a VMEbus board to verify compliance with timing specifications, the ground reference **MUST** be taken from the board's ground pin nearest the signal pin being measured, and the signal voltage **MUST** be measured on the board's connector pin.

6.4 Bus driving and receiving requirements

This subclause defines the driver and receiver specifications for all VMEbus signal lines. Table 56 lists all of the signals and shows which of the following subclauses discuss it.

6.4.1 Bus driver definitions

Totem-pole, three-state, and open-collector drivers are defined as follows:

Totem-pole
an active driver in both states which sinks current in the low state and sources current in the high state. Totem-pole drivers are used on signals having only a single driver per line (e.g., daisy-chain lines).

Three-state
similar to a totem-pole driver except that it can go to a high impedance state (drivers turned off) in addition to the low and high logic states. Three-state drivers are used for lines that can be driven by several devices at different points on the bus (e.g., address or data lines). Only one of these drivers can be active at any one time.

Rescinding
a three-state driver that is driven high before being released to a high impedance.

Open-collector
sinks current in the low state but sources no significant current in the high state. Terminating resistors on the backplane ensure that the signal line voltage rises to a high level whenever it is not driven low. Open-collector drivers are used for signal lines which can be driven by several devices simultaneously (e.g., interrupt and bus request lines).

Table 56 – Bus driving and receiving requirements

SIGNAL NAME	Subclause 6.4.2.x
A[31..1]	2
ACFAIL*	5
AM[5..0]	2
AS*	1
BBSY*	5
BCLR*	3
BERR*	5
BG[3..0]OUT*	4
BG[3..0]IN*	4
BR[3..0]*	5
D[31..0]	2
DS0*	1
DS1*	1
DTACK*	1, 5
IACK*	2, 5
IACKOUT*	4
IRQ[7..1]*	5
LWORD*	2
RETRY*	1
SYSCLK	3
SYSFAIL*	5
SYSRESET*	5
WRITE*	2

6.4.2 Driving and loading RULEs for All VMEbus lines

RULE 6.10

All VMEbus boards **MUST** provide clamping on each VMEbus signal line that they monitor to prevent negative excursions below –1.5 V.

OBSERVATION 6.4

Standard 74LSxxx and 74Fxxx devices have internal clamping diodes on their inputs that will satisfy the clamping requirement specified in RULE 6.10.

RULE 6.11

VMEbus receivers **MUST** guarantee detection of a high logic level above a threshold of 2.0 V, as shown in Figure 80.

RULE 6.12

VMEbus receivers **MUST** guarantee detection of a low logic level below a threshold of 0.8 V, as shown in Figure 80.

PERMISSION 6.2

A three-state driver MAY be used as a totem-pole driver if its output is permanently enabled.

6.4.2.1 Driving and loading RULEs for high current three-state lines (AS*, DS0*, DS1*, rescinding DTACK*, RETRY*)

RULE 6.13

IF a VMEbus board drives AS*, DS0*, DS1*, or Rescinding DTACK*

THEN its drivers for these lines **MUST** meet the following specifications:

Low state sink current	I_{OL}	>64 mA
Low state voltage	V_{OL}	<0.6 V at I_{OL} = 64 mA
High state source current	I_{OH}	>3 mA
High state voltage	V_{OH}	>2.4 V at I_{OH} = 3 mA
Minimum source current with board pin grounded	I_{OS}	>50 mA at 0 V
Maximum source current with board pin grounded	I_{OS}	<225 mA at 0 V

RULE 6.14

When drivers are turned off, VMEbus boards **MUST** limit their loading of AS*, DS0*, and DS1* to the following values:

Current sourced by board at 0.6 V, including leakage current	I_{OZL}+ I_{IL}	<450 µA
Current sunk by board at 2.4 V, including leakage current	I_{OZH}+I_{IH}	<100 µA
Total capacitive load on signal, including signal trace	C_T	<20 pF

OBSERVATION 6.17

The rescinding DTACK* has the drive characteristics specified in RULE 6.13 and the loading characteristics specified in RULE 6.22. The control of the 3 state enable for the rescinding DTACK* is specified by RULE 2.96.

OBSERVATION 6.5

The source and sink currents listed in RULEs 6.13 and 6.14 include both driver and receiver currents sourced and sunk on the board.

6.4.2.2 Driving and loading RULEs for standard three-state lines (A[31..1], D[31..0], AM[5..0], IACK*, LWORD*, WRITE*)

RULE 6.15

IF a VMEbus board drives the lines A[31..1], D[31..0], AM0[5..0], IACK*, LWORD*, or WRITE*,

THEN its drivers for these lines **MUST** meet the following specifications:

Low state sink current	I_{OL}	>48 mA
Low state voltage	V_{OL}	<0.6 V at I_{OL} = 48 mA
High state source current	I_{OH}	>3 mA
High state voltage	V_{OH}	>2.4 V at I_{OH} = 3 mA
Minimum source current with board pin grounded	I_{OS}	>50 mA at 0 V
Maximum source current with board pin grounded	I_{OS}	<225 mA at 0 V

RULE 6.16

When drivers are turned off, VMEbus boards **MUST** limit their loading of the lines A[31..1], D[31..0], AM[5..0], IACK*, LWORD*, and WRITE* to the following values:

Current sourced by board at 0.6 V, including leakage current	$I_{OZL} + I_{IL}$	<700 µA
Current sunk by board at 2.4 V, including leakage current	$I_{OZH} + I_{IH}$	<150 µA
Total capacitive load on signal, including signal trace	C_T	<20 pF

OBSERVATION 6.6

The source and sink currents specified in RULES 6.15 and 6.16 include both driver and receiver currents sourced and sunk on the board.

6.4.2.3 Driving and loading RULEs for high current totem-pole lines (SYSCLK, BCLR*)

RULE 6.17

VMEbus systems **MUST** have no more than one board driving each of the lines SYSCLK, or BCLR*. Its drivers for these lines **MUST** meet the following specifications:

Low state sink current:	I_{OL}	>64 mA
Low state voltage:	V_{OL}	<0.6 V at I_{OL} = 64 mA
High state source current:	I_{OH}	>3 mA
High state voltage:	V_{OH}	>2.4 V at I_{OH} = 3 mA
Minimum source current with board pin grounded:	I_{OS}	>50 mA at 0 V
Maximum source current with board pin grounded:	I_{OS}	<255 mA at 0 V

RULE 6.18

All VMEbus boards **MUST** limit their loading of the lines SYSCLK, and BCLR* to the following values:

Current sourced by board at 0.6 V, including leakage current	$I_{OZL} + I_{IL}$	<600 µA
Current sunk by board at 2.4 V, including leakage current	$I_{OZH} + I_{IH}$	<50 µA
Total capacitive load on signal, including signal trace, for system controllers (which have drivers)	C_T	<20 pF
Total capacitive load on signal, including signal trace, for other boards (which have no drivers)	C_T	<12 pF

OBSERVATION 6.7

The source and sink currents specified in RULEs 6.17 and 6.18 include both driver and receiver currents sourced and sunk on the board.

6.4.2.4 Driving and loading RULEs for standard totem-pole lines (BG[3..0]OUT*/BG[3..0]IN*, IACKOUT*/IACKIN*)

RULE 6.19

IF a VMEbus board drives the lines BG[3..0]OUT*/BG[3..0]IN*, or IACKOUT*/IACKIN*,

THEN its drivers for these lines **MUST** meet the following specifications:

Low state sink current	I_{OL}	>8 mA
Low state voltage	V_{OL}	<0.6 V at I_{OL} = 8 mA
High state source current	I_{OH}	>400 µA
High state voltage	V_{OH}	>2.7 V at I_{OH} = 400 µA

RULE 6.20

All VMEbus boards **MUST** limit their loading of each of the lines BG[3..0]OUT*/BG[3..0]IN*, and IACKOUT*/IACKIN* to the following values:

Current sourced by board at 0.6 V, including leakage current	$I_{OZL} + I_{IL}$	<600 µA
Current sunk by board at 2.4 V, including leakage current	$I_{OZH} + I_{IH}$	<300 µA
Total capacitive load on signal, including signal trace	C_T	<20 pF

OBSERVATION 6.8

The source and sink currents specified in RULEs 6.19 and 6.20 include both driver and receiver currents sourced and sunk on the board.

OBSERVATION 6.18

BG[x]IN* sink current has been increased to 300 µA to accommodate the Auto System Controller function (see 5.8). Standard VMEbus BGOUT drivers which provide 400 µA of source current are fully compliant to this new specification.

SUGGESTION 6.9

For Auto System Controller boards use a ≥10 kΩ to tie an open BG3IN* to logic low.

Use ≤40 µA I_{IH} receivers to receive BG3IN*.

6.4.2.5 Driving and loading RULEs for open-collector lines

(BR[3..0]*, BBSY*, IRQ[7..1]*, DTACK*, BERR*, SYSFAIL*, SYSRESET*, ACFAIL*, IACK*)

RULE 6.21

IF a VMEbus board drives the lines BR[3..0]*, BBSY*, IRQ[7..1]*, DTACK*, BERR*, SYSFAIL*, SYSRESET*, ACFAIL*, IACK*,

THEN its drivers for these lines **MUST** meet the following specifications:

Low state sink current	I_{OL}	> 48 mA
Low state voltage	V_{OL}	< 0.6 V at I_{OL} = 48 mA

OBSERVATION 6.19

The driver specification for rescinding DTACK* is specified in RULE 6.13.

RULE 6.22

All VMEbus boards **MUST** limit their loading of the lines BR[3..0]*, BBSY*, IRQ[7..1]*, DTACK*, BERR*, SYSFAIL*, SYSRESET*, ACFAIL*, IACK* to the following values:

Current sourced by board at 0.6 V, including leakage current	$I_{OZL} + I_{IL}$	<400 µA (DTACK* and BERR*) <600 µA (all others)
Current sunk by board at 2.4 V, including leakage current	$I_{OZH} + I_{IH}$	<50 µA
Total capacitive load on signal, including signal trace	C_T	<20 pF

OBSERVATION 6.9

The sink current specified in RULEs 6.21 and 6.22 includes both driver and receiver currents sourced and sunk on the board.

SUGGESTION 6.6

Since most TTL drivers do not work reliably when the +5 V d.c. power source is out of its specified range, drive SYSRESET* on a POWER MONITOR module with a driver built from a discrete high-gain small signal transistor.

6.5 Backplane signal line interconnections

The VMEbus is a high performance interface system. Its design takes into account transmission line effects on the backplane. The address and data set-up times specified in clauses 2 and 4 take into account the fact that most drivers available today do not reliably drive backplane signal lines from the low to the high level until there is a reflection from the end of the bus. Although these reflections serve a useful purpose, they cannot be excessive or ringing will result. The following paragraphs specify the backplane characteristics that achieve the desired result.

6.5.1 Termination networks

RULE 6.23a

Termination networks **MUST** be used on each end of all VMEbus signal lines except the daisy-chain lines and the Serial Bus lines.

OBSERVATION 6.10

The terminations in the VMEbus serve four purposes:

– they reduce reflections from the ends of the backplanes.
– they provide a high state pull-up for open-collector drivers.
– they restore the signal lines to the high level when three-state devices are disabled.
– they provide a standing current for the driver sink transistor to switch off, causing the signal line to rise more swiftly on positive transitions.

The Thevenin equivalent of the termination is shown in Figure 81. The voltage divider also shown provides this termination value.

OBSERVATION 6.11

IF a maximum tolerance of ±5 % is maintained on the resistor values and source voltage used in the resistor network shown in Figure 81.

THEN the circuit shown will meet the tolerances shown for the Thevenin equivalent.

OBSERVATION 6.12

The resistor network shown in Figure 81 presents its Thevenin equivalent impedance only when its +5 V source is adequately decoupled to ground by a bypass capacitor.

RECOMMENDATION 6.3

Provide a bypass capacitor with a value in the range of 0.01 µF to 0.1 µF as close as possible to the V c.c. pin of each resistor termination package.

PERMISSION 6.3

Any resistor network and voltage source MAY be used to provide the termination, as long as they provide the Thevenin equivalent shown in Figure 81.

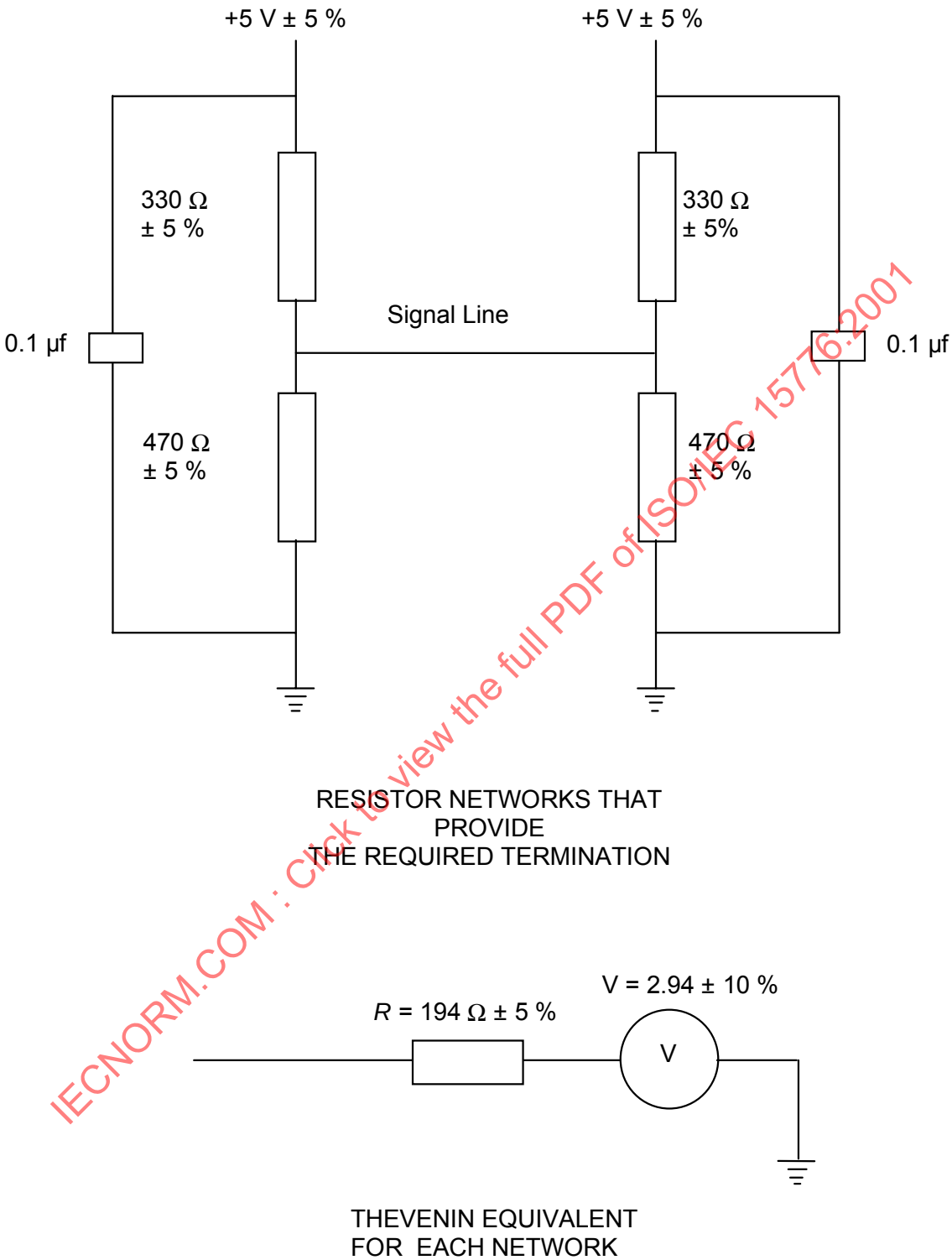


Figure 81 – Standard bus termination

6.5.2 Characteristic impedance

Each signal line in the backplane has an associated characteristic impedance Z_o. This characteristic impedance is important because discontinuities in Z_o (due to capacitive effect and loads on the bus) and mismatches between Z_o and the terminations can cause distortions of signal waveforms.[1)]

The terminations on the VMEbus signal lines reduce distortion of their signal waveforms. Although a perfect impedance match (which totally eliminates distortions due to reflections) is not maintained between the termination networks and the signal lines, it is important not to allow too great a mismatch, as might be the case if a signal line's Z_o value is too low.

The actual characteristic impedance of a backplane signal line is called the effective characteristic impedance (Z_o'), and will be lower than Z_o, due to the capacitance of plated through holes and connector pins. This additional capacitance makes Z_o' go below Z_o. Although plated-through holes are necessary to accommodate connectors, other holes should be kept to a minimum.

The backplane signal line impedance (without any boards plugged into the backplane) can be calculated with the following equation:

$$Zo' = \frac{Zo}{\sqrt{1 + Cd/Co}}$$

where

Z_o is the impedance of the microstrip line, ignoring the loading effects of plug-in pc boards, connectors, and plated-through holes;

C_d is the distributed capacitance, per unit of distance, of the plated-through holes, and backplane connectors;

C_o is the intrinsic line capacitance, per unit of distance, of the microstrip line, ignoring the loading effects of plug-in pc boards, connectors, and plated-through holes;

Z_o' is the backplane signal line impedance, including the loading effects of connectors, and plated-through holes but excluding the loading effects of plug-in PC boards.

OBSERVATION 6.13

Typical Z_o'' values for a VMEbus backplane, with no boards inserted, range from 50 Ω to 60 Ω. If this impedance is 50 Ω, or higher, it will provide satisfactory operation.

6.5.3 Additional information

RULE 6.24

All circuit traces from the 96 pin connectors to the on-board circuitry except for signals ACFAIL*, SYSRESET*, SYSFAIL*, IRQ[7..0] **MUST NOT** have a length of greater than 50.8 mm (2 in). Circuit traces from the 96 pin connectors to the on-board circuitry for signals ACFAIL*, SYSRESET*, SYSFAIL*, IRQ[7..0] **MUST NOT** have a length of greater than 101.6 mm (4 in).

OBSERVATION 6.14

IF the trace from the 96 pin connector to on-board circuitry branches,

THEN the length of each branch is added to get the total length specified in RULE 6.24.

RULE 6.25

There **MUST NOT** be more than one driver driving the SYSCLK line.

---

1) More information on characteristic impedance and backplane design can be found in the MECL System Design Handbook, Mororola, 1983.

RULE 6.26

The SYSTEM CLOCK DRIVER module **MUST** be installed in Slot 1 of the backplane.

OBSERVATION 6.15

Locating the SYSTEM CLOCK DRIVER on the board in Slot 1 minimizes the distortion of their waveforms reflected from the terminated ends of the backplane.

SUGGESTION 6.7

If actual capacitance loading values cannot be obtained from manufacturer specifications sheets, the following values can be used to estimate the total capacitive loading of a VMEbus board:

– typical capacitance of a receiver	3 pF to 5 pF
– typical capacitance of a driver	10 pF to 12 pF
– typical capacitance of a transceiver	15 pF to 18 pF
– typical capacitance of a 50,8 mm (2 in) PC trace	2 pF to 3 pF

OBSERVATION 6.16

Circuit traces which run parallel to each other, such as in a backplane, sometimes induce signal transitions in each other. This phenomenon is commonly known as crosstalk. When designing VMEbus backplanes, the spacings of lines and their position relative to ground and power planes have a large effect on the amount of cross talk observed.

SUGGESTION 6.8

Propagation delays through bus drivers depend on how heavily they are loaded and VMEbus signal lines typically represent heavy loads. This has to be taken into account when calculating worst case timing. If the manufacturer's data sheet for the driver gives a propagation delay for a 300 pF load, use that to do the worst case calculations. If the only propagation delay values are for a 30 pF load, add 10 ns to the propagation delay and 15 ns to the turn-on delay.

6.6 User defined signals

RECOMMENDATION 6.5

If a board has a 96 pin connector in its P2 location, do not allow any of the P2 pins to be driven to a voltage greater than ±15V. This reduces the likelihood of serious damage to the VMEbus system in the event that a signal trace from one of these pins is accidentally shorted to some other signal line.

6.7 Signal line drivers and terminations

This subclause summarizes the types of drivers which have to be used for each of the signal lines on the VMEbus.

In order to simplify Table 57, an abbreviated notation is used to describe the various types of drivers. The notations used are shown below:

Totem-pole (high current)	– TP HC
Totem-pole (standard)	– TP STD
Three-state (high current)	– 3 HC
Three-state (standard)	– 3 STD
Open-collector	– OC

For detailed specifications, see 6.4.

Table 57 – Bus driver summary

SIGNAL MNEMONIC	SIGNAL NAME	DRIVER TYPE	BUSED AND TERMINATED
A[31..1] (31 lines)	ADDRESS BUS	3 STD	YES
ACFAIL*	AC POWER FAILURE	OC	YES
AM[5..0] (6 lines)	ADDRESS MODIFIER	3 STD	YES
AS*	ADDRESS STROBE	3 HC	YES
BBSY*	BUS BUSY	OC	YES
BCLR*	BUS CLEAR	TP HC	YES
BERR*	BUS ERROR	OC	YES
BG[3..0]IN* BG[3..0]OUT* (Daisy-chain)	BUS GRANT DAISY-CHAIN	TP STD	NO
BR[3..0]* (4 lines)	BUS REQUEST	OC	YES
D[31..0] (32 lines)	DATA BUS	3 STD	YES
DS0*-DS1* (2 lines)	DATA STROBES	3 HC	YES
DTACK*	DATA TRANSFER ACKNOWLEDGE	OC 3 HC	YES
IACK*	INTERRUPT ACKNOWLEDGE	3 STD or OC	YES
IACKIN*/IACKOUT* Daisy-chain)	INTERRUPT ACKNOWLEDGE DAISY-CHAIN	TP STD	NO
IRQ[7..1]* (7 lines)	INTERRUPT REQUEST	OC	YES
LWORD*	LONGWORD	3 STD	YES
RETRY*	RETRY	3 HC	YES
SER_A	SERIAL BUS	Note 1	Note 2
SER_B	SERIAL BUS	Note 1	Note 2
SYSCLK	SYSTEM CLOCK	TP HC	YES
SYSFAIL*	SYSTEM FAILURE	OC	YES
SYSRESET*	SYSTEM RESET	OC	YES
WRITE*	WRITE	3 STD	YES

NOTE 1 Driver types are defined by the serial bus specification.
NOTE 2 Terminations are defined by the serial bus specification.

7 Mechanical specifications

7.1 Introduction

The information provided in this clause ensures that the VMEbus board assemblies, backplanes, subracks, and associated mechanical accessories are dimensionally compatible.

The mechanical dimensions conform to IEC 60297-1, IEC 60297-3, IEC 60297-4 and IEC 60603-2. The electrical characteristics for VMEbus connectors, as specified in clauses 5 and 6, supersede IEC 60603-2 where they differ.

Two connector types are permitted in this specification, one for 96 pins configurations and one for 160 pin[1] configurations. For 96 pin configurations use the IEC 60603-2 which describes a family of connector types which are identified by labels of the form: 60603-2-IEC-xxxxxx-xxx. All of the P1/J1 and P2/J2 connectors used on VMEbus boards and backplanes are members of the 60603-2-IEC family. In this chapter, the label 60603-2-IEC-xxxxxx-xxx is used when referring to all of these connector types as a group. The label 60603-2-IEC-C096Mx-xxx is used when referring to the male connector types within these families which are used on VMEbus boards. 60603-2-IEC-C096Fx-xxx is used when referring to the female connector types which are used on VMEbus backplanes.

The drawings in this specification show the 160 pin family of connectors. These connectors have five rows of contacts. Use the center three rows when applying the drawings to 96 pin family of connectors.

Figure 82 is a front view of a 19-in wide subrack and shows how single height and double height VMEbus boards can be mixed in a single subrack. Boards are inserted into the subrack from the front, in a vertical plane with the component face of the board on the right.

PERMISSION 7.1

A VMEbus system MAY be composed of single height boards, double height boards, or a mixture of both.

RULE 7.1

Single height VMEbus subracks **MUST** have a single J1 backplane.

RULE 7.2

Double height VMEbus subracks **MUST** have

1) a J1 backplane mounted in the upper portion of the subrack,

OR

2) a J1 and a J2 backplane, with the J1 backplane mounted in the upper portion and the J2 backplane mounted in the lower portion.

OR

3) a double height backplane which provides both J1 and J2 connectors.

RULE 7.3

VMEbus backplanes **MUST NOT** have more than 21 slots.

---

[1] At the time this document was printed the 160 pin connector was being standardized in the IEC 61076-4 family of connectors rather than the IEC 60603-2 family. In this clause when the 160 pin connector is referenced, the reader should refer to the IEC 61076 standard rather than IEC 60603-2. Additionally, when the text refers to the connector described in IEC 60603 in a general way the reader should also include an IEC 61076 connector.