Contents

Figures

Tables

Educational Services

Futurebus+ Handbook for Digital Systems

EC-H1363-41

Version Information: First printing. This is an evolving document.

February 1992

Digital Equipment Corporation

February 1992

The postpaid Reader's Comments form at the end of this document requests your critical evaluation to assist in preparing future documentation.

The document was prepared using VAX DOCUMENT, Version 2.0.

About this Handbook

Preface

Purpose

This handbook is a design guide for developers of Futurebus+ modules intended for use in Digital Equipment Corporation systems. Module designers, system integrators, Original Equipment Manufacturers, and end users of Digital products should find this handbook useful.

The guide includes engineering details about system architecture, hardware interface, and board layout. The purpose of this handbook is to provide a reference to many of the necessary concepts, rules, and guidelines to ensure interoperation of Futurebus+ systems and modules to maximize customer satisfaction.

Digital has endorsed Futurebus+ Profile B as the standard I/O bus for its server and multi-user systems. Profile B is specifically defined as a high-performance, general purpose computer I/O bus. Futurebus+ meets or exceeds the I/O bus requirements for these systems for many years to come, thus stabilizing Digital I/O bus architecture well into the future.

Audience

This handbook is intended to address the needs of two sets of readers (see Figure 1).

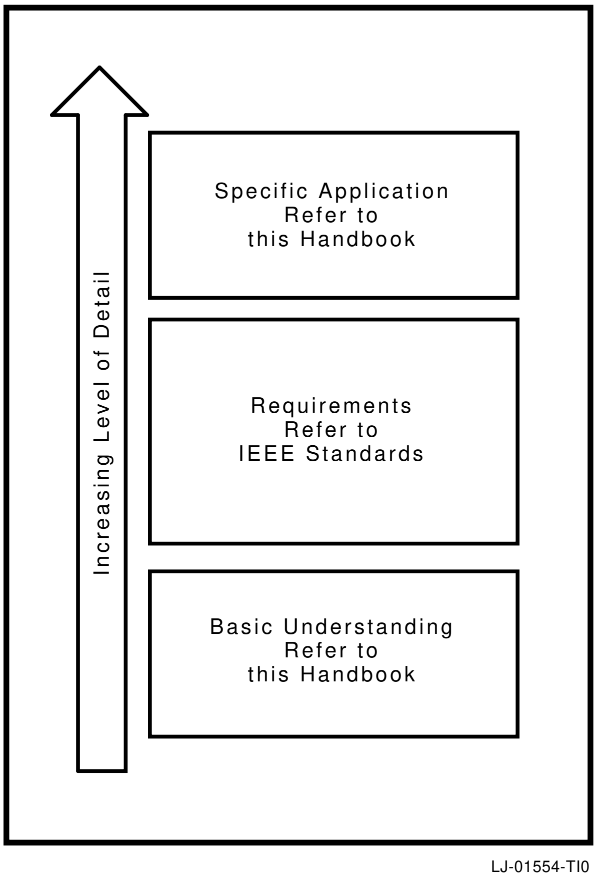
The first set of readers are those just beginning to explore how the Futurebus+ works and how to use it. This handbook contains information that can help them understand what the Futurebus+ is and what it can do for them. The handbook will also introduce these readers to the Institute of Electrical and Electronic Engineers, Inc. (IEEE) standards that control the Futurebus+ and provide valuable references within each chapter to related standards.

Continued on next page

The second set of readers are the designers of Futurebus+ options or systems. For these readers, this handbook provides:

- Detailed information to complement the IEEE standards
- Specific information on the approach that Digital takes to implement Futurebus+ systems and options
- Solutions that Digital has found to some of the problems facing designers

Figure 1 Intended Audience



Continued on next page

Organization

The information in this handbook is more practical than theoretical. It enhances the material found in the IEEE 896 Futurebus+ standard and provides detailed descriptions of the following:

- Overall hardware and system architecture
- Bus hardware interface
- Diagnostics
- Fault management
- Examples

Information Not Included

This handbook does not cover all of the details contained in the IEEE standards (for example, specific logic equations); designers should consult the latest versions of the standards before starting a design. Information about ordering the related IEEE standards is at the end of this chapter.

Continued on next page

Chapter Organization

This handbook is organized as follows:

- Chapter 1, Introduction, briefly describes the Futurebus+, the background of the profiles used to specify the electrical, mechanical, environmental, and functional requirements, and the implementation of Profile B in particular.
- Chapter 2, Architecture, provides information about the architecture of Futurebus+ Profile B systems, which includes address space, bridging between Profile B and other buses, Profile B transactions, and the Futurebus+ Control Status Registers (CSRs).
- Chapter 3, Profile B Logical Layer, describes the individual lines that comprise the Futurebus+ and the various operations that employ these lines.
- Chapter 4, Physical Layer, describes the electrical and mechanical aspects of Futurebus+ Profile B.
- Chapter 5, System Test, provides the guidelines for diagnostic testing to be independent of the instruction set architecture, system configuration, and operating system.
- Chapter 6, Fault Management, describes the conventions for recording state upon detection of errors, recovery by retry, and reporting errors to the next level in the system.
- Appendix A, CRC-16 Algorithm, provides an example of a CRC algorithm.
- Appendix B, CSR Architecture ROM Directory Entries, provides a summary of the ROM definition values defined in the IEEE 1212 CSR Architecture standard.
- The Glossary of Terms contains definitions of terms used in this handbook.

Continued on next page

Handbook Conventions

This handbook uses the following words according to their definitions in the IEEE 896 standards:

May	This word is used to indicate flexibility of choice with no implied preference.
Shall	This word is used to indicate a mandatory requirement. Designers must implement all such mandatory requirements to ensure interoperability with other IEEE 896 compliant designs.
Should	This word is used to indicate flexibility of choice with a strongly preferred implementation. The phrase *it is recommended* is used interchangeably with the word *should*.

This handbook uses the following prefix conventions for numbers:

0b	Precedes a binary number.
0x	Precedes a hexadecimal number.
None	The number is given in decimal form.

Chapter 1
Introduction

Futurebus+ Overview

Futurebus+ Definition

The Institute of Electrical and Electonic Engineers, Inc (IEEE) 896 Futurebus+ is a comprehensive backplane bus standard. This standard and related documents describe the logical, physical, and environmental requirements for building Futurebus+ systems and peripherals.

Continued on next page

Futurebus+ Function

The Futurebus+ is designed for cost effective extensibility for a wide variety of applications. It attempts to be capable of the highest possible transfer rate consistent with the technology available at the time modules are designed, while ensuring compatibility with all other modules designed to this standard.

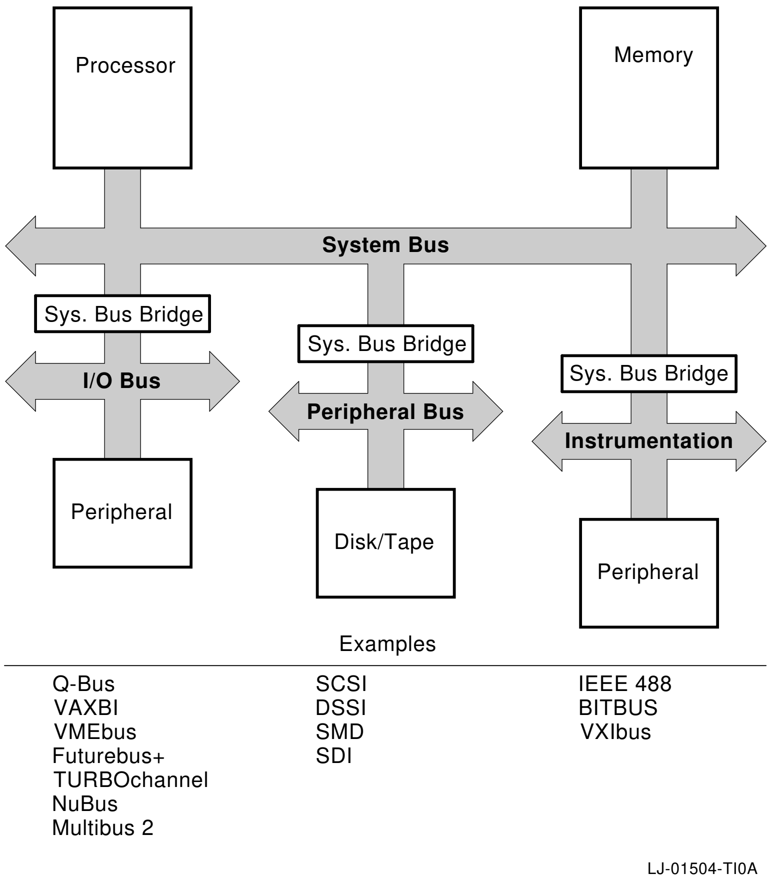
Not all Futurebus+ functions are needed in every application environment: A Futurebus+ Application Environment profile is created by selecting options from a palette of Futurebus+ logical, physical, and environmental specifications to meet specific application requirements. Complete multiprocessor, cache-coherent systems, and simple I/O interconnect schemes can be built upon the rich Futurebus+ protocol set.

Continued on next page

Futurebus+ Overview, Continued

Figure 1–1 illustrates the two-bus topology and taxonomy.

Figure 1–1 Generic Bus Topology and Taxonomy



Profile B

Background

Modern computer systems use a multiple bus hierarchy for a variety of reasons. The system bus that interconnects processors and memory is inextricably related to the processor architecture. This bus is subject to change with every improvement in processor technology. The system bus tends to be very complicated and supports a wide variety of features, primarily for performance reasons.

Systems must also be able to access, interpret, and manipulate their environments. This requires the ability to customize the computer system with application-specific devices. The definition of these interfaces will remain unchanged for long periods of time; therefore, they need a consistent way to interface to successive generations of computers. This requirement led to the development of an I/O bus that is connected to the system bus with a bridge. Although the system bus is constantly being modified, the definition of the I/O bus will remain unchanged for many years.

Profile Functions

The function of a profile is to ensure compatibility within the class of products that adhere to its rules by restricting the choices from the full Futurebus+ palette to an internally consistent subset. A profile does this in three independent dimensions:

- At the logical layer, the legal bus signals, their timing and interactions, the supported protocol functions, the architectural behavior, and the hierarchical behavior when choices are permitted are all defined to ensure functionality and interoperability.
- At the physical layer, electrical and mechanical connectivity, parameters impacting signal integrity, power distribution, mechanical dimensions and their tolerances are specified to ensure interoperability, interconnectivity, and intermateability.

Continued on next page

Profile Functions (continued)

- Environmental specifications are provided to both target specific classes of applications and to manage the parameters that control reliability.

Profile B Implementation

IEEE 896 Futurebus+ Profile B is a subset of the full Futurebus+ bus protocol. It targets an I/O bus for high-performance, general purpose computer systems in an open-office, computer room, or unattended environments.

The logical features of Profile B on Digital systems are:

- Central arbitration
- 32/64-bit address width
- 32/64-bit data width
- Simple transaction set
- Futurebus+ CSR set
- Simple interrupt scheme
- High performance Direct Memory Access (DMA) architecture

The physical features of Profile B are based on the electrical requirements of backplane transceiver logic (BTL) transceivers, IEEE 1194.1, and the mechanical specifications of IEEE 1301 and 1303.1. This includes 12 SU x 12 SU card cage and module dimensions and I/O through the front panel only.

The environmental class of Profile B is specified by IEEE P1156.1, Class 4.

Features and Benefits of Futurebus+ Profile B

Open Standard

IEEE 896 Futurebus+ represents a consensus of many groups throughout the computer industry. As a true open standard, IEEE 896 is fully endorsed by systems suppliers, peripheral developers, semiconductor suppliers, and companies providing development tools. The broad-based acceptance of the standard ensures that a rich set of Futurebus+ tools, products, and components will be readily available.

Performance

The throughput of IEEE 896 Futurebus+ exceeds the I/O bus requirements of the modern computer technology by several orders of magnitude. Through technological advancements and the introduction of packet mode transactions, the performance of the bus will continue to improve as needed. This capability allows computer manufacturers who adopt the Futurebus+ as their I/O bus to maintain a consistent interface over many years. This is a significant advantage to the developer who wishes to exploit advancements in computer technology while maintaining application interfaces over time.

Integrity

Data integrity of the Futurebus+ exceeds that of any industry standard bus ever developed. The electrical characteristics of IEEE 896 Profile B are well understood. Positive noise margins can be guaranteed within the design specifications. The protocol support for error detection and reporting on a byte basis provides for a reliable interface.

Continued on next page

Features and Benefits of Futurebus+ Profile B, Continued

Digital Support

Digital Equipment Corporation provides a variety of services for the developers of IEEE 896 Futurebus+ Profile B peripherals. These services include:

- A developers kit that provides detailed design information
- Telephone support for technical issues
- Design verification centers
- Inclusion in Digital product literature

Industry Support

Vendors within the computer industry (including Digital) provide additional support services:

- Compatible protocol interface chips that are available through commercial semiconductor vendors
- IEEE Computer Society Bus Architecture Standards Committee (BASC)
- VME Bus International Trade Association (VITA)

Related Documents

IEEE Standards

Readers who would like more information about the Futurebus+ should procure the IEEE standards listed in Table 1–1.

The three principal standards referenced in this handbook are described briefly in this section.

Table 1–1 Related IEEE Standards

Number	Title
IEEE 896.1	Futurebus+ Logical Specifications
IEEE 896.2	Futurebus+ Physical Layer and Profile Specifications
IEEE P896.3	Recommended Practices
IEEE 1212	Control and Status Register Architecture
IEEE P1212.1	DMA Framework Architecture
IEEE 1301.1	Detailed Standard for a Metric Equipment Practice for Microcomputers Using 2 mm Connectors and Convection Cooling
IEEE P1156.1	Environmental Specifications for Computer Modules
IEEE 1194.1	Electrical Characteristics of Backplane Transceiver Logic Interface Circuits

IEEE 896.1

This standard defines the following:

- Technology independent logical specifications
- Bus signal definitions
- Bus arbitration
- Parallel protocols
- Control and Status Registers (CSRs)

Continued on next page

Related Documents, Continued

IEEE 896.2

This standard identifies and defines Futurebus+ profiles. Futurebus+ Profile B (which Digital implements) is defined in this standard. Although all profiles comply to IEEE 896.1, each profile may have its own logical, physical, and environmental requirements and definitions.

IEEE P896.3

This standard is intended to characterize a variety of systems topics as they relate to Futurebus+. When used in conjunction with other IEEE standards, the details are available to develop modular, open-architecture based systems fulfilling user needs across a wide computing spectrum.

Ordering IEEE Standards

The IEEE standards may be ordered from the following address:

The Institute of Electrical and Electronic Engineers, Inc.
345 East 47th Street
New York, New York 10017

Implementation Techniques

In some instances, this handbook offers specific implementation details to interpret the standard. Digital encourages the use of these implementation techniques to further promote interoperation of systems and option modules.

Chapter 2
Architecture

Overview

Chapter Overview

This chapter provides the following information about the architecture of Profile B systems.

- Definition of terms
- The Futurebus+ address space and how this space maps to various system address spaces
- Profile B transactions
- Bridging between Profile B and other buses including system buses
- The Futurebus+ Control Status Registers (CSRs)

Continued on next page

Overview, Continued

Key Terms

The following key terms are used in this chapter.

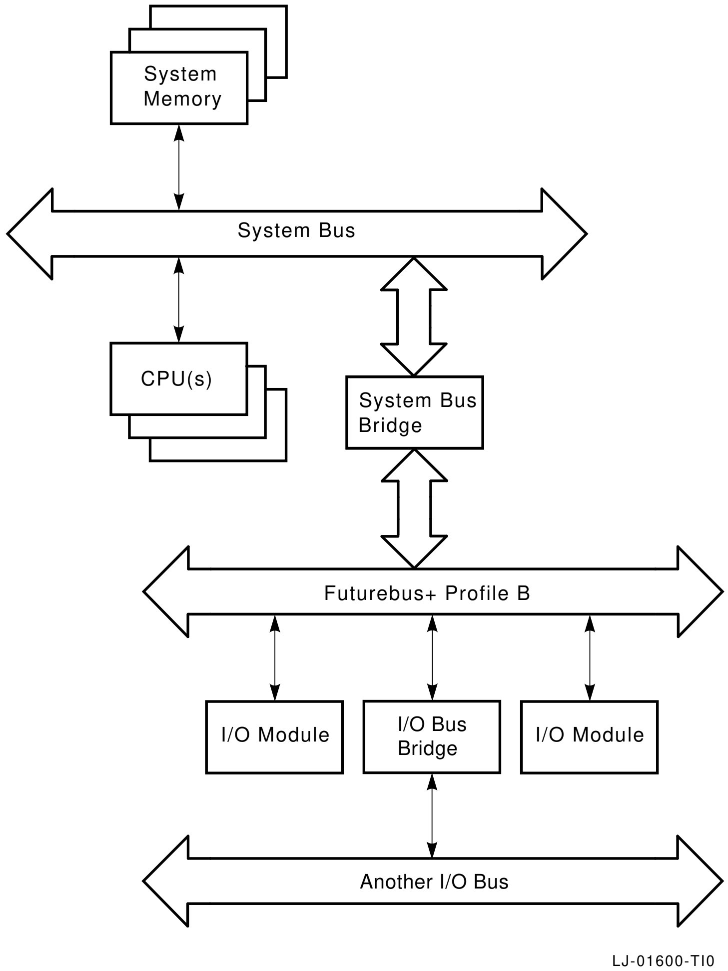
Term	Meaning
System architecture	The design of the central processor that includes the instruction set, addressing modes, I/O, and the way memory is managed
System bus bridge	A unit architecture that connects the system bus to the Futurebus+ Profile B I/O bus, another I/O bus, or both.
I/O Bus bridge	A unit architecture that connects two or more I/O buses. The I/O bus bridge provides signal and protocol translation from one bus to another.
I/O Module architecture	A unit architecture that connects Futurebus+ Profile B to an I/O port
DMA Architecture	An architecture that supports independent access of I/O modules to memory for commands, data, and status

System Architecture

Definition

Profile B is a general purpose, high-performance I/O bus. The systems are typically configured as shown in Figure 2–1.

Figure 2–1 System Block Diagram



Continued on next page

System Components

A typical Digital computer system is composed of several interacting components.

- CPU

  One or more central processing units that usually include cache memory.

- System memory

  The storage for instructions and data that is processed by the CPUs.

- System bus

  The internal path that interconnects the CPUs and system memory. The system bus also provides access to the I/O equipment of the system by way of the system bus bridge.

- System bus bridge

  The device that provides protocol conversion, address mapping, and so forth, between the system bus and one or more I/O buses.

- Profile B

  The particular I/O bus described in this handbook.

- Profile B I/O module

  A device used to connect I/O equipment to Profile B.

- I/O Bus bridge

  A bridge used to provide protocol conversion, address mapping, and so forth, between the Futurebus+ and other I/O buses (for example, VMEbus, Q–bus, and so on.)

Continued on next page

Optional Features

The IEEE Futurebus+ Profile B standard specifies several optional features. The optional features not supported on Digital systems at this time include the following:

- Arbitrated message passing (see the section ''The System at Initialization'' in this chapter)
- Distributed arbitration
- Packet mode
- Serial bus
- 128-Bit data path
- Use of TAG bits
- Live insertion

Modules, Nodes, and Boards

Modules

A module is the entire entity consisting of one or more physical printed wiring boards (PWBs) that is connected to the Futurebus+ by a single set of transceivers (see Figure 2–2). A module may contain zero, one, or two nodes.

A module with zero nodes can not contain CSRs; therefore, the module acts as a passive bus *watcher* or *listener*. For example, a module that is used to interface to a logic analyzer is a zero-node module.

Node

A node is an entity controlled by a single set of CSRs. The CSRs of one node function independently of actions of the CSRs on any other node.

All current Digital Profile B modules implement only one node. This does not mean that all Digital modules have a single back-end port (such as an Ethernet, SCSI bus, or serial port); it means that all of the back-end ports are controlled by a single set of Futurebus+ architectural CSRs. A node may contain (control) more than one unit. One or more sets of unit CSRs may be present to control these units.

Boards

A board is the physical entity—the PWB. One or more boards constitute a module.

Range of Complexity

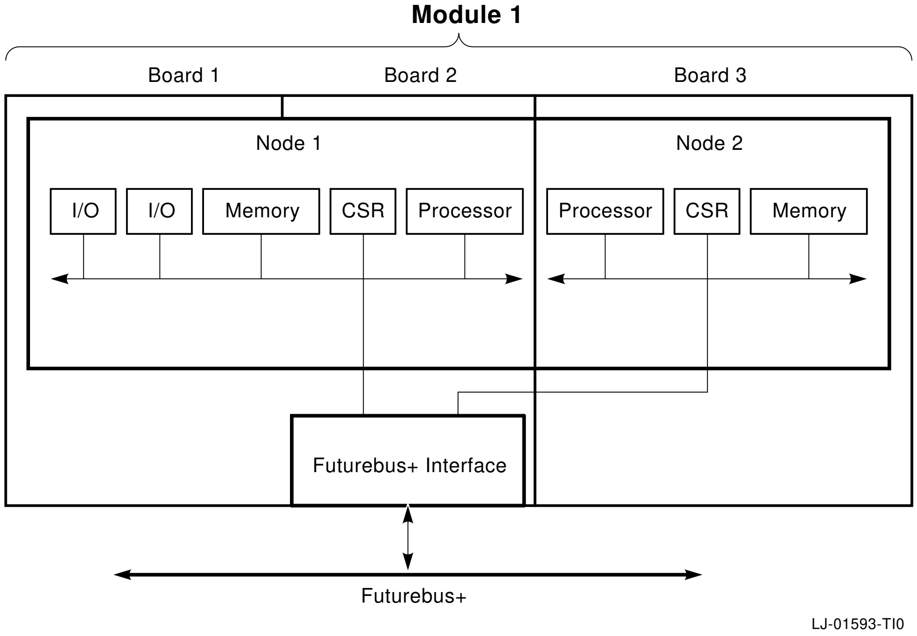
Profile B modules can range from being very complex to very simple. A complex module can have two nodes per module, several PWBs, a full set of CSRs, and various back port connections. A simple module can act as a slave only, contain a minimum CSR implementation, and have a very simple back port. Digital systems support modules of all complexity levels.

Continued on next page

Example

The following figure illustrates a single Futurebus+ module.

Figure 2–2 Futurebus+ Modules, Nodes, and Boards



This module spans three physical PWBs. As required by the definition of a module, there is only one set of Futurebus+ transceivers (shown on Board 2 in Figure 2–2). The module also contains the maximum two permissible nodes. As required by the definition of a node, both nodes shown in the example are substantially independent, sharing only their Futurebus+ interface.

Futurebus+ Address Mapping

Address Widths

Futurebus+ supports two address widths.

- 32 Bits (A32)
- 64 Bits (A64)

All Futurebus+ modules must support A32 addressing. As an option, modules may support A64 addressing.

CSR Addressing

CSRs in initial node space can only be accessed using A32 addressing (see the main section ‘‘CSR Addressing’’ in this chapter). This requirement allows interoperability of all modules in all Futurebus+ systems regardless of whether the individual modules support the optional A64 addressing.

A32 Address Mapping

Using A32 addressing, the 2^{32} (4096 megabytes [MB]) of address space is divided into two main regions.

- 256 MB of CSR space
- 3840 MB of remaining space

The further division of the 256 MB CSR space is further described in the main ‘‘CSR Addressing’’ section of this chapter.

In Digital systems, the 3840 MB of remaining space is used to access system memory by way of the system bus bridge. This is further explained in the ‘‘System Address Mapping’’ section of this chapter.

Continued on next page

A64 Address Mapping

Using A64 addressing, the entire 2^{64} bytes (16 million terabytes) of address space is treated as some form of non-CSR address space. The module CSRs in initial node space are *not* accessible. In Digital systems, a portion of the 2^{64} address space will be used to access system memory through the system bus bridge. This is further explained in the ‘‘System Address Mapping’’ section of this chapter.

When issuing addresses lower than 2^{32} - 2^{28} (0xF0000000), A32 addressing (not A64 addressing) must be used. This allows all modules to see these transactions, regardless of whether each module is A64-capable.

System Address Mapping

Overview

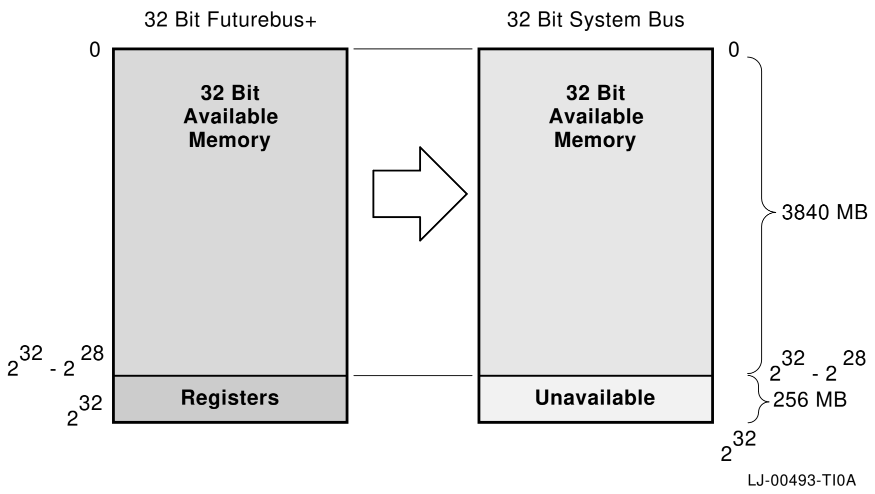
System configurations can have differing amounts of memory and address space. Figure 2–3, Figure 2–4, and Figure 2–5 illustrate how the Futurebus+ address space is mapped to the memory space on Digital systems.

The Futurebus+ defines a 32-bit and a 64-bit address space. This address space is mapped to system memory space through the system bus bridge.

A32 Futurebus+ Addressing to 32-Bit System Bus Addressing

Some Digital systems support up to 32 bits of system address space. On these systems, I/O modules need only use A32 transactions to access system memory. The address mapping is shown in Figure 2–3.

Figure 2–3 32-Bit Futurebus+ to 32-Bit System Bus Mapping



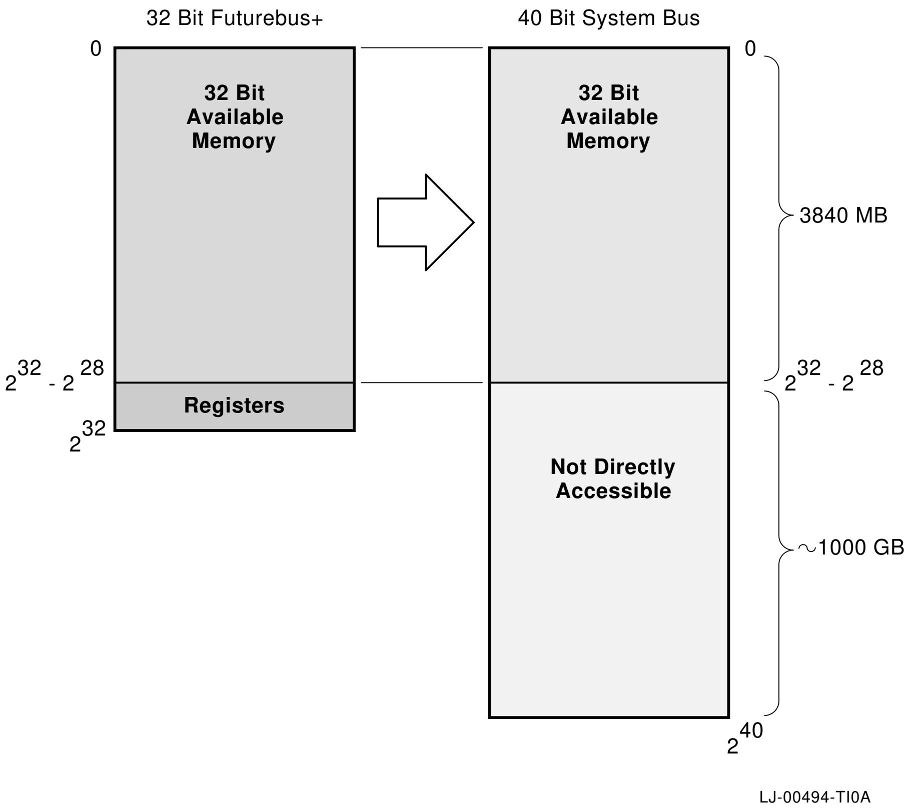
Continued on next page

A32 Futurebus+ Addressing to 40-Bit System Bus Addressing

Higher performance Digital systems support as many as 40 bits of addressing to system memory.

Current Digital system bridges do not provide paged I/O address mapping; therefore, modules that support only A32 transactions map directly to system memory (see Figure 2–4). The I/O nodes that support only A32 transaction widths cannot directly access more than 3840 MB (2^{32} - 2^{28} bytes) of memory. If user buffers need to be allocated in addresses above this limit, the software must allocate a temporary buffer in the 32-bit available memory and copy it to the user buffer when the DMA is complete.

Figure 2–4 32-Bit Futurebus+ to 40-Bit System Bus Mapping



Continued on next page

64-Bit Address Capability (A64 Addressing)

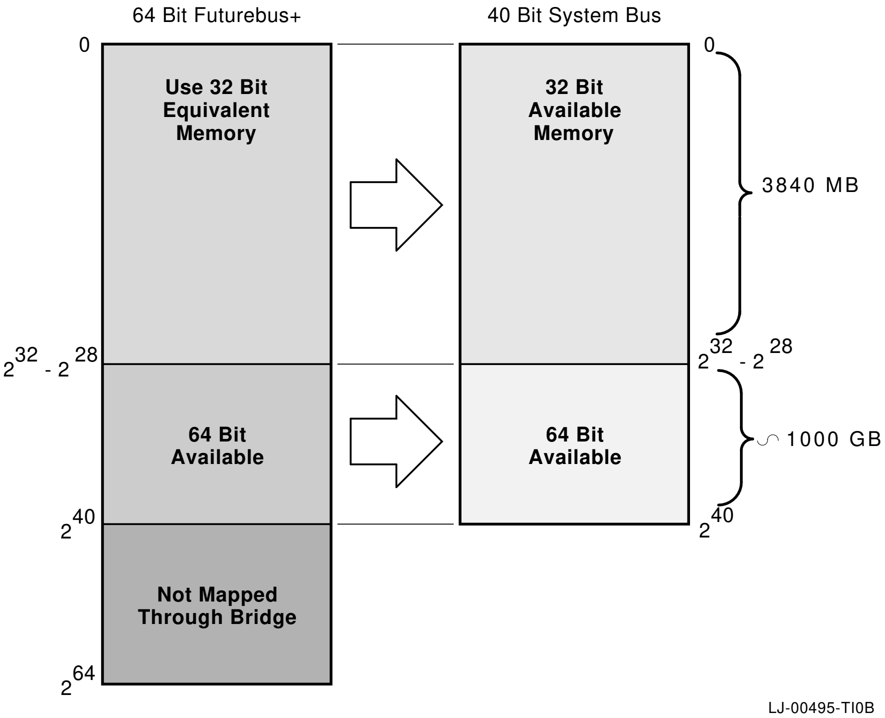
Nodes on the Futurebus+ that support A64 capability must also support A32 capability. Addresses that map to the lower 3840 MB (2^{32} - 2^{28} bytes) of memory must use the A32 equivalent address. Addresses from 3840 MB (2^{32} - 2^{28} to 2^{40}) must use A64 addresses. The initial node space CSRs on the Futurebus+ are only accessible using A32 addressing; therefore, the A64 capable I/O nodes must be able to distinguish between CSR addresses that must use A32 and memory addresses (that map to the same addresses as the CSRs), which must use A64 addressing.

A64 Futurebus+ Addressing to 40-Bit System Bus Addressing

Figure 2–5 shows a mapping configuration in which system memory is available to 64-bit capable I/O nodes. In Digital high-performance systems, addresses above the 2^{40} limit are not forwarded to the system bus. Devices on the Futurebus+ that want to use local memory, such as frame buffers, may use these addresses.

Continued on next page

Figure 2–5 64-Bit Futurebus+ to 40-Bit System Bus Mapping



View from the System

The previous portions of this section have described the view *inwards* from the Futurebus+ to the system memory bus. This section describes the view *outwards* from the CPU to the Futurebus+.

Continued on next page

Mailboxes

Modern CPUs are very fast compared to the speed of random read and write operations on an I/O bus; therefore, it is undesirable to allow the CPU to directly read or write to locations on the I/O bus. In the time that the CPU would spend waiting for the return of read data or the confirmation of the write, the CPU could have executed tens or hundreds of instructions.

To alleviate this problem, Digital CPUs will access the Futurebus+ using intelligence built into the system bus bridges. The CPU will build a data structure in its memory indicating the details of the Futurebus+ transactions to be performed. The system bus bridge will then perform the actual transactions, updating the in-memory data structures with results and status.

In effect, the CPU is performing all Futurebus+ operations as split transactions.

A32 and A64, Byte Maskings, Etc.

Using various bits in the data structure, the CPU is able to command the details of the individual Futurebus+ transactions. For more information about the implementation of mailboxes on Digital computers, see the documentation for the system bus bridge for particular systems. System software hides most of these details from user-level software.

CSR Addressing

Purpose

The individual nodes on Futurebus+ modules are controlled from the Futurebus+ by reading and writing a series of CSRs contained on each node.

Addressing CSRs

These CSRs all exist in a reserved 256 MB section of the A32 space of the Futurebus+. The 256 MB section is further subdivided, as shown in Figure 2–6 and Figure 2–7, with each Futurebus+ node allocated 4 kB.

In a theoretically maximum Futurebus+ configuration, there can be 1023 individual Futurebus+ buses, each with 62 nodes, yielding 1023x62 = 63,426 nodes.

The Futurebus+ standard allows for the possibility of CSR accesses being *forwarded* from one physical Futurebus+ to another. This capability may or may not exist in a given actual system. In addition, the Futurebus+ standard allows as many as 62 nodes to exist on a single Futurebus+. Profile B allows only a maximum of 14 modules, which limits the maximum number of nodes per Futurebus+ bus to 28.

Bus Number

Each node can respond to either of two address ranges.

- The range corresponding to bus 1023 (the local bus)
- The range corresponding to the bus identified in the NODE_IDS CSR

  This value initially defaults to 1023, the local bus.

This implies that a node can always be accessed from the local bus using the address range corresponding to bus 1023. If there is more than one Futurebus+ bus segment, this node may also be accessed by its bus segment number.

Continued on next page

Figure 2–6 CSR Address Mapping

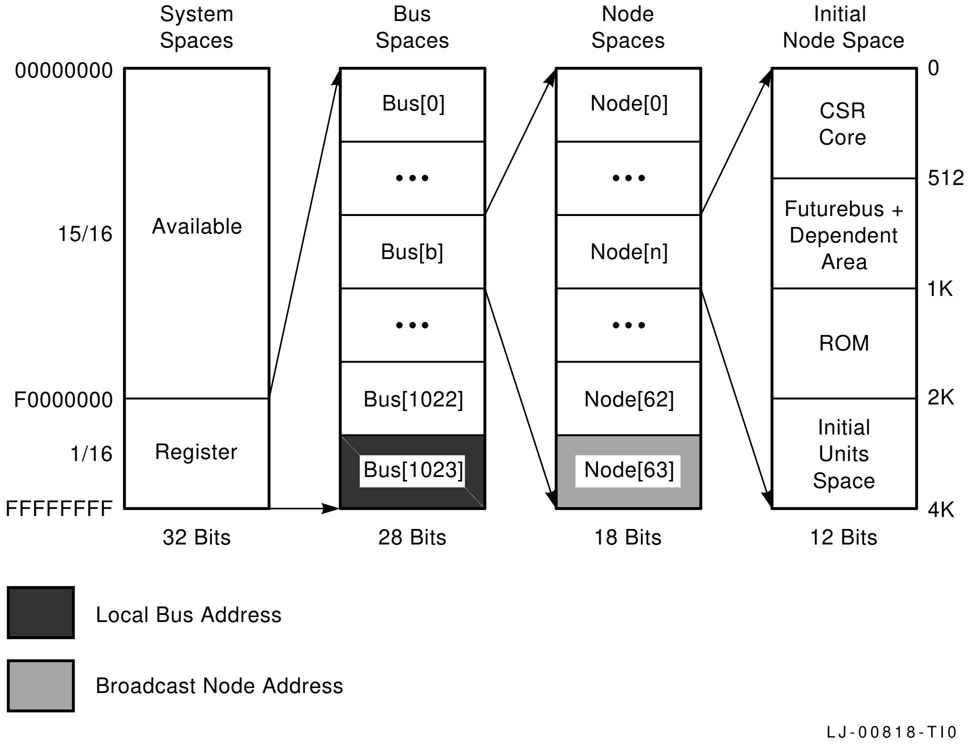
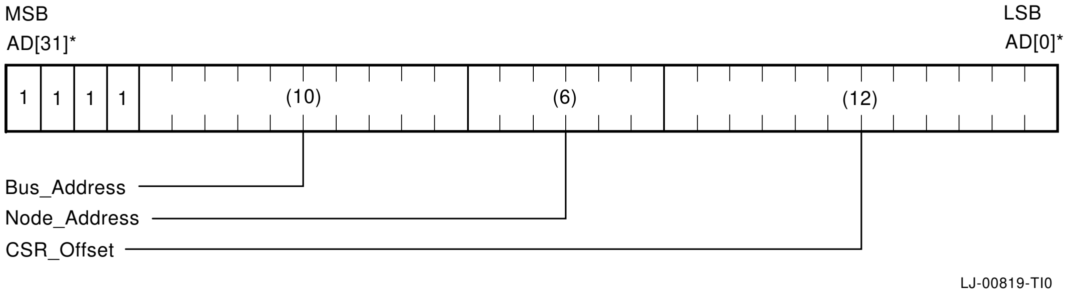


Figure 2–7 CSR Addressing on the Futurebus+



Continued on next page

No Broadcast

Futurebus+ Profile B does not support Node 63 (broadcast) addressing. Profile B modules may ignore Node 63 addresses.

Allocation of the 4 kB Node CSR Space

There are four regions within the 4 kB space that are allocated to each node:

- Core CSRs
- Futurebus+ dependent CSRs
- ROM CSRs
- Initial Unit Space CSRs

Core CSRs

Core CSRs are defined by the IEEE 1212, CSR Architecture standard. In this area, the Core CSRs are implemented in a bus-independent fashion; that is, they are not specific to the Futurebus+.

Futurebus+ Dependent CSRs

Futurebus+ dependent CSRs are defined by the IEEE 896.2 standard as being specific to the Futurebus+. A space is also reserved in this area for vendor-dependent applications.

Continued on next page

ROM CSRs

The ROM area contains ROM and ROM-like locations, which give fixed (invariant) information about the node. This includes capabilities, name, and so forth.

Initial Units Space

This space is intended to contain CSRs relating to individual subunits (for example, disk drives) that are controlled by the node.

Further Details

The individual CSRs within the Core CSR, Futurebus+ Dependent CSR, and ROM CSR regions are described in detail later in this chapter.

Arbitration, Bus Mastership, and Bus Tenure

Overview

Only one Futurebus+ module may control the Futurebus+ parallel bus at a time. The module in control of the parallel bus is referred to as the *master* or *bus master*. The time during which a given module is master is referred to as the *bus tenure* of that module.

Arbitration

When a module needs to send or receive data from system memory or another module, it must first gain mastership of the bus. More than one module can request bus tenure at any one time; therefore, the arbitration process is used to restrict bus tenure to only one module at a time. The arbitration process for Profile B is central arbitration. (Distributed arbitration is not supported on Digital systems at this time.)

With the central arbiter scheme, each module that requests use of the bus issues a request to a system central arbiter. The arbiter, operating under a fair priority scheme, issues a grant to one module at a time. When a module is granted use of the bus, it must wait for the current bus master to end its tenure. The newly designated master (the master-elect) can then gain tenure over the bus. Once a module has gained tenure of the bus, it is free to initiate one or more transactions.

To allow fair access to the bus in a timely manner, the central arbiter can issue a *preemption* to the current bus master if another bus device is making a higher priority request. The current bus master, upon recognition of preemption, must finish its current transaction and release its request for the bus. The central arbiter will then grant bus tenure to the higher priority module.

Continued on next page

Arbitration, Bus Mastership, and Bus Tenure, Continued

Arbitration Details

As previously mentioned, arbitration in Digital systems is *fair*. Averaged over time, all modules have equal access to the bus. The Futurebus+ standards, however, allow other arbitration schemes to be imposed.

An additional detail of arbitration involves the RQ0* and RQ1* lines. In some systems, these two lines allow requests to use the Futurebus+ at two different priority levels. All transactions except Read Response must use RQ0*, the lower of the two priority levels. Read Response transactions may use RQ1*, the higher of the two priority levels. Some systems (including some Digital systems) treat RQ0* and RQ1* requests identically and do not distinguish between the two priority levels.

Transactions

Overview

Each basic operation using the Futurebus+ data parallel bus is referred to as a *bus transaction*. Bus transactions consist of operations such as Read Unlocked, Write Unlocked, and Read Response.

Split and Connected Transactions

Some bus transactions are part of higher-level system transactions. Each system transaction can be viewed as consisting of two parts: a request and a response.

Connected Transaction

If the request and the response take place in a single bus transaction, they are referred to as a *connected transaction*. (Some buses refer to such connected transactions as *unpended*).

Split Transaction

If the request takes place in one transaction and the response takes place in a later, separate bus transaction, they are referred to as a *split transaction*. (Some buses refer to such split transactions as *pended* transactions because the operation is left pending for a period of time.) System transactions are split so that valuable bus time is not wasted waiting for a slow resource (such as memory) to respond.

On Profile B, only the two read transactions (Read Unlocked and Read Partial) can be split. When these transactions are split, they are referred to as Read Unlocked request and Read Partial request in those sections of this document that refer to timing diagrams. In each case, the data is returned in a later Read Response bus transaction.

The overall Futurebus+ standard allows either the master or the slave to initiate the splitting of a transaction. However, Profile B allows only the slave device to initiate the split.

Transaction Phases

Overview

From the frame of reference of an I/O module, each bus transaction takes place in three distinct phases.

- Connection
- Data
- Disconnection

Connection Phase

During this phase, the bus master establishes its logical connection with the intended slave device.

If the connection is established, the bus master passes command information to the slave device and receives status information back from the slave device. If data is to be passed between the master and the slave, the master will proceed to the data transfer phase. If there is no data (at the time) to be passed between the master and the slave, the master will proceed to the disconnection phase.

If the connection fails to be established, the master proceeds directly to the disconnection phase.

Data Phase

During this phase, data is transferred to or from a slave.

Continued on next page

Disconnection Phase

During this phase, the master terminates the transaction and disconnects from the slave. The master can then end its bus tenure by proceeding to the idle phase or the master can begin another bus transaction by proceeding directly to the connection phase.

During the disconnection phase, the master can also provide information that is useful to the slave, either immediately or at a later time.

In the case of a Read Unlocked or Read Partial transaction that is split at the request of the slave, the disconnection information from the module making the read request contains data that the responder will use later to identify and reconnect to the requester.

When the responder becomes bus master and issues a Read Response transaction back to the original requester, the responder will use the disconnection information to indicate the overall success or failure of the split read request.

Finally, a DMA protocol, *MORE*, uses the disconnection information to identify particular streams of DMA data that is prefetched or post-written. This protocol is described later in this chapter.

Profile B Transactions and Commands

Overview

Profile B is intended for use as an I/O bus; therefore, it does not need to support the entire set of transactions defined in the IEEE 896.1 standard.

The transactions in this section are the only transactions supported by Profile B.

Read Unlocked

This transaction is used to transfer data from the slave to the master. The slave may convert a connected Read Unlocked transaction into a split transaction in which the read transaction becomes an address-only transaction.

Read Partial

This transaction is used to transfer data from the slave to the master. The master specifies to the slave that it will evaluate only some of the bytes transferred within the specified data path width. The slave may convert this transaction into a split transaction.

Address Only

The system software or device drivers are not capable of initiating an Address Only transaction on Digital systems. This form of a transaction can result on the Futurebus+ when software has initiated a Read Unlocked or Read Partial transaction and the transaction is split by the slave, or an error occurs during the connection phase.

Continued on next page

Read Response

This transaction is the response to a previous Read Unlocked or Read Partial transaction that was split by the original responder.

The transaction includes a data phase in which data are written to the original requesting module. Either the compelled or the packet protocol can be used for this transaction. The global identification and transaction ID of the requester is transmitted in the connection phase; status is transmitted in the disconnection phase. There is only one Read Response returned for a single split read request. That is, the response cannot be further split.

Write Unlocked

This transaction is used to transfer data to the slave from the master. The slave shall not convert a Write Unlocked transaction into a split transaction.

Write Partial

This transaction is used to transfer data to the slave from the master. The master specifies to the slave which of the transferred bytes should be evaluated by the slave. The slave shall not convert this transaction into a split transaction.

Profile B Transaction Command Qualifiers

Overview

Profile B transaction commands can be modified (qualified) as described in the commands in this section.

Address Width

As a general rule, device drivers will use only A32 (32-bit) addresses when initiating transactions on the Futurebus+ because all Initial Unit Space CSRs on Futurebus+ are in A32 address space.

It is possible for I/O devices that support A64 (64-bit) addressing and implement extended units space or extended memory space to have CSRs in A64 address space. When this is necessary, the configuration software allocates address space for these devices in A64 address space. Devices that require this kind of configuration include graphics frame buffers, array processors, and memory boards.

Data Width

As a general rule, device drivers will use only 32-bit (D32) data when initiating transactions on the bus. At system configuration time, this is an absolute rule: The initial node space CSRs accessed to determine the character of a node and to set capability enables are only required to support D32 data.

The exception to this rule is Initial Unit Space CSRs (part of a port architecture) that have been designed to require atomic D64 transactions. Therefore, generic operating system mailbox interface routines need to support D64 data width.

Busy Bus Resources and Busy Retries

Overview

Bus transactions may access a resource that is not capable of responding at the moment. For example, a read may be requested of a device that has a full read command queue. In these cases, the slave device will return a *Busy* status to the master.

Generally, the master device will be designed in such a way that it will automatically retry the operation that received the Busy status. How often and how many times the master will retry the operation is controlled by values loaded into several CSRs within the master.

Control Registers

The BUSY_RETRY_DELAY CSR contains the value of the time delay between the retries of a busy resource by the master. The delay time is specified in units of 2^{-32} seconds ($\approx$233 ps).

The BUSY_RETRY_COUNTER CSR contains two fields: one contains a counter that counts retries; the other contains a threshold value beyond which a BUSY_RETRY error is declared. If this threshold is set to zero, the master will not attempt any retries of a busy resource.

Data Buffer Alignment on the Futurebus+

Overview

The optimum transfers on the Futurebus+ are always naturally aligned. That is, data of length n bytes must always be transferred starting at an address that is evenly divisible by n. For example, a D32 transfer (4 bytes per data beat) must always start with an address that is evenly divisible by 4 (and therefore ends in 0b00).

The Futurebus+, however, is defined as allowing transfers from any arbitrary byte. This means that for unaligned transfers, special actions must be taken.

For read operations, the bus master can employ either of the following two strategies:

- Issue a Read Partial bus transaction.
- Issue a Read Unlocked bus transaction and ignore the data bytes that are not needed.

As a practical reality, the second option is faster; there is no need to use the first data beat to pass the byte lane masks to the slave.

For write operations, the bus master must employ the Write Partial bus transaction.

This section describes three typical transactions and the strategies to use to accommodate them on the Futurebus+.

Continued on next page

Unaligned Addresses

If an I/O module is given a starting address that is not aligned, the I/O module must issue a Write Partial transaction for the first data beat to mask out data that should not be written. In the case of a read, it should read the entire data length from memory and internally mask out the bytes not needed. This eliminates the need to do a Read Partial transaction for one data beat, which is inefficient, particularly for split operations.

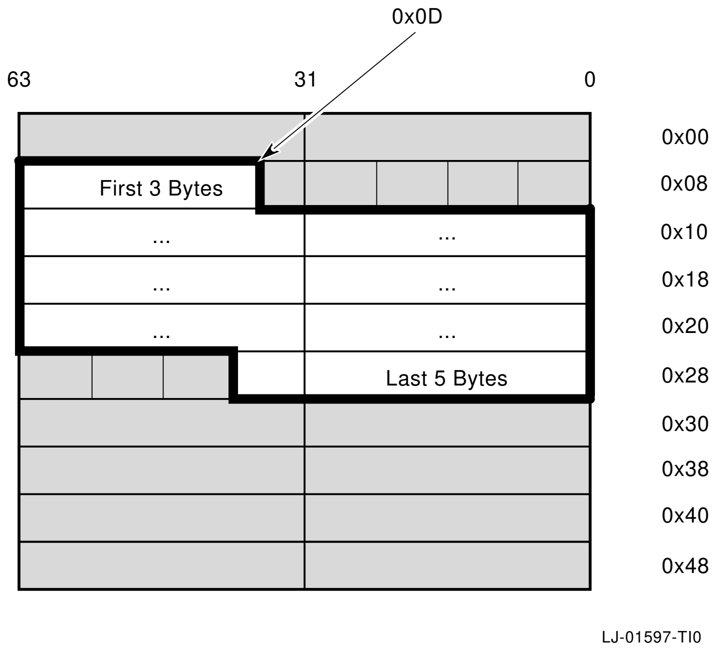
As an example, assume a buffer is created in memory starting at address location 0x0D, and that the following conditions apply:

- The I/O module is using D64 transfers and was set up to use DL[2..0] equal to 0b011 (4 data beats).
- The length of the total DMA transfer is 32 bytes.

Continued on next page

The buffer will appear in memory, as shown in Figure 2–8.

Figure 2–8 Sample Buffer (1)



Continued on next page

DMA Write Addressing Example 1

For a DMA write, the I/O module would first issue a Write Partial transaction to address 0x08, with lane disable L[4..0] asserted, and DL[2..0] set to 0b001 (1 data beat). If the module acting as the master asserts DL[2..0] to 0b011 (4 data beats), the bridge acting as the slave will assert ED* on the first data beat, signaling that the master should proceed to the disconnection phase and issue another transaction to continue the DMA.

The next write transaction would start at address 0x10 and have DL[2..0] set to 0b011 (4 data beats). The data length DL[2..0] is also the address boundary (calculated by multiplying data width by data length) over which a transaction cannot cross; therefore, the master must disconnect after the second data beat (32-byte alignment).

The master continues the DMA transfer by issuing another write starting at address 0x20, with DL[2..0] equal to 0b011 (4 data beats). There is only one data beat before the last partial 8-byte transfer; therefore, the master enters the disconnection phase after one data beat.

The master completes the transaction by issuing a Write Partial to address 0x28, with lane disable L[7..5] asserted to deselect the upper 3 bytes of the 8-byte transaction.

Continued on next page

DMA Read Addressing Example 1

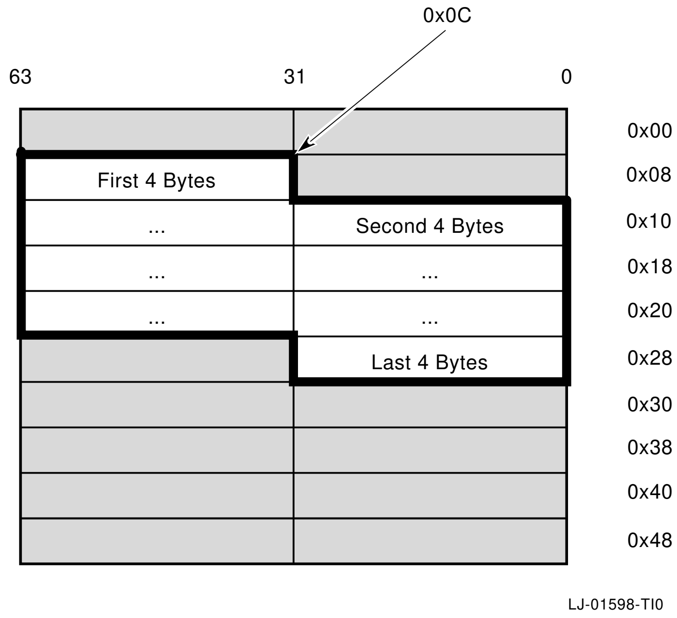
For a DMA read, the I/O module should issue a read transaction starting at address 0x08, and use a DL[2..0] of 0b011 (4 data beats), as set up by the configuration process. Internally, the I/O module must mask the lower 5 bytes of the first 8-byte transfer because they do not contain valid data for this DMA. The I/O module must disconnect after 3 data beats (32-byte alignment address).

The I/O module can then complete the DMA transfer by issuing another read transaction starting at address 0x20. The I/O module must mask the upper 3 bytes of the last 8-byte transfer because they do not contain valid data for this transfer.

4-Byte Aligned Addresses

As a second example, consider the buffer shown in Figure 2–9. In this example, the situation is improved from the buffer shown in Figure 2–8, in that this buffer is aligned to a 4-byte boundary. Assume all other conditions remain unchanged.

Figure 2–9 Sample Buffer (2)



Continued on next page

DMA Write Addressing Example 2

In this case, the transactions can be similar to the previous example, except the lane disables will be L[3..0] for the first Write Partial and L[7..4] for the last Write Partial. However, a slightly more complex I/O module could recognize the fact that the transaction is not optimally aligned for D64 transfers and instead use D32 transfers. In this case the I/O module would issue a write transaction with a data length of 0b100 (8 data beats), to maintain the 32-byte alignment set up by the configuration process. The first transaction would begin at address 0x0C, and proceed for 5 data beats to bring the address up to the 32-byte alignment. The next transaction would start at address 0x20, and proceed for 3 data beats.

DMA Read Addressing Example 2

For devices that do not use the Read Partial transaction, a read transaction would be identical to the previous example.

Lengthy Aligned Transfer Example

As a final example, consider a 232 byte transfer starting at address 0x10, and using a DL[2..0] of 0b100 (8 data beats), with a data width of 64 bits. The data placed in memory appears as shown in Figure 2–10.

DMA Write Addressing Example 3

The I/O module would issue the first write transaction starting at address 0x10 and continuing for 6 data beats. The second and third transactions start at the next 64-byte aligned addresses and continue for 8 data beats. The last transaction starts at address 0xC0 and continues for 7 data beats.

DMA Read Addressing Example 3

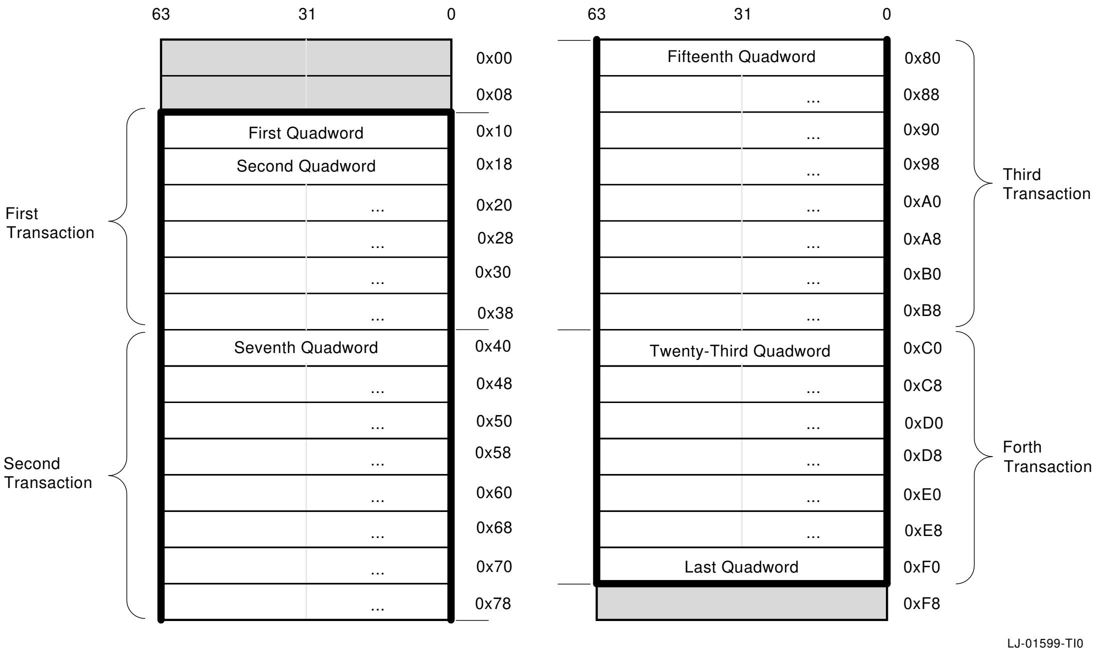
A read transaction would be identical to the write transaction.

Continued on next page

Byte Packing and Unpacking

If the I/O module can receive data on any arbitrary byte alignment and send data to any other arbitrary byte alignment, the I/O module must be able to shift and pack bytes as necessary to match the bus alignment and data length field used.

Figure 2–10 Sample Buffer (3)



Interrupts

Futurebus+ Interrupt Mechanisms

Unlike some buses, the Futurebus+ does not employ any dedicated hardware mechanism to pass interrupt requests from one module to another, such as interrupt request or acknowledge lines. The interrupts are passed by causing the interrupting module to write to a predetermined register on the module(s) to be interrupted.

There are two distinct Futurebus+ interrupt mechanisms as follows:

- Use of the INTERRUPT_TARGET CSR (not recommended by Digital)
- Use of CSRs in unit space (unit-specific interrupt mechanism)

Profile B Interrupt Mechanism

Profile B implements the unit-specific interrupt mechanism defined in Chapter 3 of IEEE 896.2 standard. During initialization, the I/O module will be passed a CSR address for every interrupt priority level supported by the system, and a data value representing the interrupt vector that will be used for posting interrupts. To post an interrupt, the module will issue a Write Unlocked transaction to the appropriate CSR in the system bridge. The system bridge has a finite storage capacity for interrupts; therefore, the I/O node can receive a Busy response to the CSR write. In this case, the I/O node must relinquish the bus and wait the duration of time stored in its BUSY_RETRY_DELAY CSR before retrying the transaction. The I/O node must retry the transaction until the transaction succeeds or the retry count equals the value in its BUSY_RETRY_THRESHOLD CSR. It must then report an error and stop retrying that transaction.

An I/O module should not consider a busy interrupt attempt as an error or exception condition until it times out on retries. Normal DMA operation should be allowed to continue during interrupt retries. An error should be reported only after the busy retry threshold has been reached.

Continued on next page

Interrupt Addresses

The system bridge on Digital systems will implement four CSR addresses for targeting interrupts. Table 2–1 can be used to determine the interrupt addresses.

Table 2–1 Interrupt Addresses

Address Offset into Bridge CSR Space	Interrupt Level
AD[11..0] = 800	IPL 14
AD[11..0] = 804	IPL 15
AD[11..0] = 808	IPL 16
AD[11..0] = 80C	IPL 17

Each I/O node should use two interrupt levels: one level for DMA completion, and a higher level for error interrupts. A node should make attempts at reducing the number of completion interrupts generated by allowing one interrupt to signal the completion of more than one DMA. Because the system bridge implements the interrupt CSRs in FIFO queues, there is a possibility of having a queue at a particular level being temporarily full.

In the unlikely event of a Busy timeout on an interrupt write, the I/O node should generate an error interrupt to a CSR representing the higher error IPL, if possible.

System Initialization

Overview

Futurebus+ systems require software to perform various initialization and configuration procedures. These procedures ensure that all of the modules on a particular Futurebus+ system are operating in concert, using similar values for timeouts, and so forth. The IEEE standards define the entity responsible for executing this initialization and configuration software as the *monarch*.

Monarch Selection

A Futurebus+ system may have only one monarch at a time. In Digital systems, the system CPU that connects to the system bus bridge is always designated as the monarch.

The initialization and configuration software is a combination of console and diagnostic firmware and operating system software.

Reset Assertion

The reset line (RE*) is a wire-OR'd bus signal that allows one or many modules to assert RE* at the same time. All I/O modules are expected to time the duration of RE* assertion; its interpretation depends upon how long RE* is asserted. The timing of this signal starts when the RE* signal transitions from released to asserted.

Continued on next page

Types of Reset Pulses

Pulses that are received on the RE* line fall into one of the following four categories:

- 0.001 to 1.999 ms— live insertion notification
- 2 to 30 ms—bus initialization
- 30 to 500 ms– system reset
- Longer than 500 ms after power has reached regulation—the RE* line is considered to be stuck asserted

  This may occur normally during initial powerup. If this occurs after powerup has completed, the RE* line is considered to be stuck asserted.

Modules that drive the RE* line have significantly tighter timing specifications on the pulses they generate.

- 0.001 to 0.129 ms—live insertion notification
- 4 to 12 ms bus initialize.
- 100 to 200 ms—system reset

  During powerup, this pulse can be extended to 500 ms (until the module can respond to CSR accesses).

Continued on next page

Power Up

By the time the 2.1 V power has reached 40% of regulation, the RE* line will be driven to the asserted state on the backplane. On power up as with system reset, all modules on the Futurebus+ initiate built-in self-test (BIST) diagnostics.

These power-up diagnostics are the *initialization* type as defined by the TEST_START CSR of the node. Results of the initialization diagnostics are recorded in the TEST_STATUS CSR.

Each module will release RE* when they are able to respond to CSR access. For Power Up Reset, this must be within 500 ms; BIST is allowed to take as long as 10 seconds to complete. As part of the module reset, all CSRs must be set to their default values.

A read reference to an existing address is acknowledged when the target node asserts the *Selected* status or the *Busy* status line (if BUSY_RETRY_COUNTER is disabled, the bridge reports a BUSY_RETRY_THRESHOLD_EXCEEDED error). After BIST has completed, Busy status will not be expected as a response to a transaction targeted at node CSR space. All defect-free devices must complete BIST in less than ten seconds after RE* is released. The return of either a Busy or Selected status is an indication that a node exists at that target address.

Continued on next page

Bus Initialization

A bus initialization begins after the occurrence of a 2 to 30 ms pulse on the RE* line. This event causes the bus interface logic on all Futurebus+ modules to reset. All modules will stay aligned to the bus (AI* asserted).

System Reset

System reset begins after the occurrence of a 100 to 200 ms pulse on the RE* line. This event causes a full reset of logic on all of the modules receiving the RE* line. BIST is then initiated by all modules.

Each module will release RE* when they are able to respond to CSR access.

BIST is allowed to take as long as 10 seconds to complete. As part of the module reset, all CSRs must be set to their default values.

Live Insertion

The live insertion pulse is when the RE* line is asserted for 0.001 to 0.129 ms. As soon as possible, the current bus master should conclude any on-going bus transaction and not begin a new transaction. As the bus becomes quiescent, the newly inserted module can synchronize (align) itself to the current state of the various Futurebus+ synchronization lines.

Continued on next page

Power Fail

All modules that need to be informed of a power fail event will monitor the arbitration bus. After the system bridge has sent the power fail message, all voltages will remain within regulated values for at least 4 ms.

Node Directed Reset

After the system has been initialized, the CPU will load the operating system and begin a system boot. At this time, the system may perform a node directed reset on each node using the RESET_START CSR. The node will then perform a complete node reset and will run BIST. All nodes will stay aligned to the bus during this process.

If the software driver detects a fatal error associated with the node, the system will perform a node directed reset on the node.

System Configuration Software

Sizing the System

The initial configuration of Profile B I/O devices on Digital systems is split between the console code and the device drivers. Following successful completion of the system bus bridge diagnostics, the console configures the various registers on the system bus bridge and then probes the Futurebus+ to determine what each device is and where each device is located. The probing of each node is begun when all nodes on the bus have released the reset line RE*.

Note that the IEEE Futurebus+ standard allows up to two nodes on each module (sides 0 and 1). Modules that have only one node are required to place the node on side 0.

An empty slot will not acknowledge either node address of the module.

Setting Configuration at Powerup

The configuration routine performs the following operations on each module after powerup, system reset, and bus initialization:

- Check TEST_STATUS CSR for successful completion of initialization and Initialization tests.
- Read MODULE_VENDOR_ID and MODULE_SW_VERSION to determine the device driver associated with the module.

  Concatenate MODULE_VENDOR_ID and MODULE_SW_VERSION to form a unique number that relates a device to a driver. The IEEE 1212 CSR standard supports other ROM entries for this purpose if the module has more than one type of node or unit.
- Write the BUS_PROPAGATION_DELAY CSR in each node to properly set the glitch filters.
- Read the MODULE_LOGICAL_CAPABILITY ROM CSR of each node to determine the module capabilities for address and data width.

Continued on next page

Setting Configuration at Powerup (continued)

- Read the NODE_CAPABILITIES_EXT ROM CSR of each node to determine the node address and data width capabilities.
- Read the NODE_CAPABILITIES ROM CSR of each node to verify that the node is capable of extended tests.
- Write the NODE_IDS CSR to update the bus number, if applicable (the default value is 1023).
- Write the SPLIT_TIMEOUT, TRANSACTION_TIMEOUT, BUSY_RETRY_DELAY, BUSY_RETRY_COUNTER, ERROR_RETRY_DELAY, and ERROR_RETRY_COUNTER CSRs to the correct value for the system.
- Write the LOGICAL_MODULE_CONTROL CSR to set 64_BIT_ADDRESS bit, DATA_WIDTH field, MASTER_ENABLE bit, and COMPELLED_DATA_LENGTH field (PARITY_REPORT_ENABLE bit clear). There is no need to write the LOGICAL_COMMON_CONTROL CSR on Profile B modules.
- If the node has implemented extended tests (as determined earlier by the read of the NODE_CAPABILITIES ROM), then the ARGUMENT CSRs will be written with the starting address of a 4-kbyte block of memory for use by extended diagnostics.
- Write the TEST_START CSR to initiate the default extended test suite.
- Check TEST_STATUS for successful completion of extended tests.
- Enable parity error reporting on the node by rewriting the LOGICAL_MODULE_CONTROL CSR to set PARITY_REPORT_ENABLE, 64_BIT_ADDRESS bit, DATA_WIDTH field, MASTER_ENABLE bit, and COMPELLED_DATA_LENGTH field.

The device driver is responsible for configuring any MEMORY_BASE, MEMORY_BOUNDS, UNIT_BASE, UNIT_BOUNDS and any Initial Unit Space CSRs.

Power Down

Signaling

Power failures on the Futurebus+ are signaled using the arbitration bus, a portion of the Futurebus+ that is separate from the data parallel bus. The arbitration bus is used by Futurebus+ for two purposes.

- Distributed arbitration (not currently supported by Digital)
- Passing of special messages

The mechanism used in both cases is essentially the same: Each module competing for the arbitration bus presents a binary-encoded numeric value to the bus and then examines the resulting wire-OR'd value. If a module recognizes the presence of a higher-numbered value, that module removes its own value. Eventually, the arbitration bus stabilizes with just one module driving the bus–the module with the highest-numbered value.

When modules are using the arbitration bus to perform distributed arbitration, the modules present unique, relatively low-numbered values to the bus.

When a module wishes to indicate an impending power failure, it presents a relatively high-numbered value to the bus. Arbitration values 0x60 through 0x7F are used to indicate power failure. Generally, the system bus bridge provides the power failure indication from the system power supplies.

Continued on next page

Reception

Any module wishing to receive power-fail notification must be able to receive and decode the power failure arbitrated message(s). System power will be maintained for a minimum of 4 ms after transmission of the message.

Cache Coherence

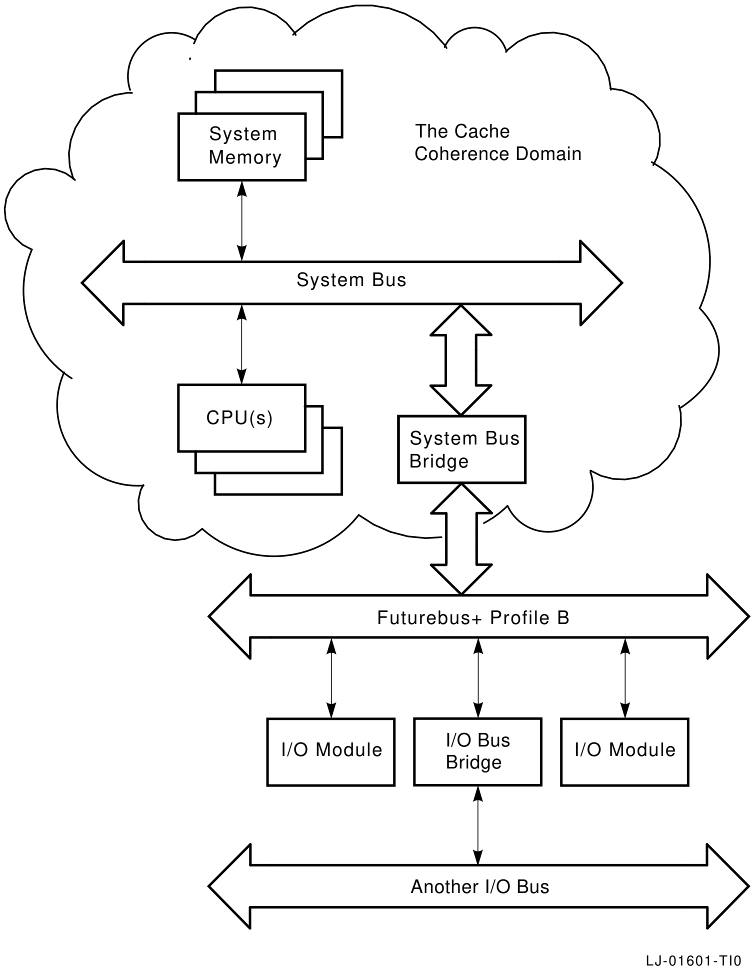
Caches and Data Coherence

Cache memories are high-speed memories used to hold *caches* of valuable data (data that the system believes will be of use in the near future).

Cache memories may be placed within any of the blocks within the *cache coherence domain* as deemed necessary by the system designers. (Generally, cache memories are placed near the point of use of the individual data items: CPU caches store instructions and data that will be used by the CPU; the system bus bridge may cache data that will be used by the I/O devices, and so forth.) The system designers implement the necessary protocols to ensure that, in some way, all parts of the system within the cache coherence domain see all data in a consistent way (see Figure 2–11). No part of the system within the cache coherence domain is allowed to see old copies of the data.

Continued on next page

Figure 2–11 Cache Coherence Domain



Continued on next page

Cache Coherence in Profile B

As defined by Profile B, the system bus bridge forms part of the boundary of the cache coherence domain. That is, if a module on the Futurebus+ implements a cache, the system *will not* take any automatic action to ensure that the data cached by that module remains consistent with the data present within the confines of the cache coherence domain.

Implications for the System Bus Bridge

To simplify the design of a cache coherence system, the system bus is often designed to support only a limited number of transaction types and sizes. Often, if the caches within the system store data in groups of a certain fixed number of bytes (referred to as the cache line size), the system bus will be designed to support only transactions of that size. Because the Futurebus+ supports a wide variety of transaction types and sizes, the system bus bridge must contain logic to accommodate this discrepancy.

Methods

For operations that read from system memory (or anywhere within the cache coherence domain), the system bus bridge may be forced to mask off some of the data bytes provided by the system bus before providing a smaller number of bytes to the Futurebus+.

For operations that write to system memory (or anywhere within the cache coherence domain), the system bus bridge may be forced to perform a read-modify-write cycle on the system bus.

In addition, for operations in either direction, the system bus bridge may need to convert multiple Futurebus+ data beats or bus transactions into a single system bus transaction or vice-versa.

System Bus Bridge Architecture

Bridge Definition

The term bridge is used to denote a unit architecture that connects two buses. A bridge can be one module with a connection to both buses, or it can be two modules, each installed in a separate bus backplane and connected to each other by a cable. The function of the bridge is to translate and map transactions on one bus to the appropriate transactions on the other bus (refer to Figure 2–1 for a block diagram of bridges on Digital systems).

System Bus Bridge Function

System bus bridges on Digital systems connect the Futurebus+ to the system memory bus. In this configuration, the bridge assumes that I/O generated DMA traffic and interrupts travel from the I/O bus to the system bus. Current Digital system bus bridges generate only single beat, 4-byte (usually CSR) or 8-byte reads and writes to addresses on the Futurebus+. The bridge is transparent to I/O modules. The system bridge must also adhere to the cache coherence protocol on the system bus because the cache domain does not extend to the I/O bus.

Splitting and Nonsplitting Bridges

Bridges have the option of allowing split or connected read transactions. For split transactions, the I/O device issues a Read Unlocked (or Read Partial) transaction that is split by the bridge acting as a slave during the connection phase. When the bridge receives the data from memory, the bridge arbitrates for the bus and issues a Read Response transaction. The I/O module must decode the AD[31..0] data in the connection phase of the Read Response transaction to determine if the response is its own.

Because of internal optimizations, certain system bus bridges that split Read Unlocked or Read Partial requests will deal with the detection of nonexistent memory errors (on the system bus) by never issuing a Read Response transaction on the Futurebus+. From the perspective of the Futurebus+ master that requested the original read, this will then appear as a split timeout error.

Continued on next page

Splitting and Nonsplitting Bridges (continued)

Additional details about the exact nature of the error can be obtained by interrogating the system bus bridge.

Optimizing Bus Throughput

Bridges with long memory latency will split read transactions to optimize for overall bus throughput. Splitting read transactions allows multiple I/O devices to issue transactions overlapped with accesses to system memory by the bridge. As a bus master, all I/O modules must support both connected and split reads.

When issuing a read transaction, an I/O module cannot tell if a read transaction will be split until the end of the connection phase.

A bridge that splits read transactions will contain enough transaction buffers to avoid becoming a bottleneck for I/O traffic.

Connected Write Transaction

On Futurebus+ Profile B I/O buses, write transactions must be connected; read transactions can be either connected or split. Once a write transaction enters the disconnection phase, the I/O module must assume the write will be completed correctly by the bridge. A higher level of verification can be initiated at the driver level to acknowledge the correct completion of a write transaction.

Continued on next page

Ordering Rules for the Bridge Designer

The bridge designer shall ensure that correct read/write ordering is maintained.

In the absence of the MORE prefetching and buffering protocol, the following rules apply to preserve read/write ordering. (See ‘‘Use of MORE’’ in the ‘‘I/O DMA Block Architecture’’ section of this chapter.)

- Write transactions from the same node must be completed in the order issued on the Futurebus+.
- Read transactions from the same node must be issued to memory in the order they were issued on the Futurebus+.
- A read following a write to the same address from the same node must read the previously written data.
- A read preceding a write from the same node must read the old data.
- A DMA completion interrupt from a node must not be made visible on the system bus before all preceding write transaction from that same node are made visible in memory, or any preceding read transaction from that same node are issued to memory.

Ordering Rules for I/O Module Designer

Designers of I/O modules must be aware of the following conditions involving bridges that split read transactions:

- Read Responses from multiple outstanding split read transactions can be returned in any order.
- An interrupt issued after a split Read Unlocked or Read Partial transaction can complete before the Read Response is returned.
- A CSR read or write from the system bus is independent of any reads or writes from the I/O bus and can complete in any order relative to I/O initiated transactions.

Continued on next page

Deadlock and Starvation

The bridge and the I/O module designers must be aware of the conditions that can cause deadlock and starvation. Starvation is a condition where a node fails to make forward progress while trying to complete its transaction and eventually times out. Deadlock is a condition where one or more nodes cannot complete or initiate a transaction until some previous condition has been satisfied but that condition cannot be. Failure to properly control the flow of Futurebus+ transactions can result in starvation or deadlock conditions.

Deadlock Avoidance Design Rules

The I/O module must perform the following actions:

- Accept and process CSR reads or writes while waiting for a Read Response.
- Split a CSR read, or store the CSR write data to free the Futurebus+, if the I/O node cannot access its internal CSR due to an internal bus conflict. The I/O node can then issue the Read Response.
- Be capable always of processing at least one read or write transaction as a slave. The module can then issue a busy status to a second CSR read or write. The bridge will relinquish tenure and free the bus for other transactions.

Continued on next page

Issuing Transactions on the System Bus

Just as an I/O device issues transactions on the Futurebus+, the bridge issues transactions on the system bus. The system bus may have a different data width and may use different transaction types than the Futurebus+. The bridge translates transactions from the Futurebus+ defined set to transaction types defined on the system bus.

Digital system bus address space is in a cache coherence domain; therefore, transactions on the system bus must be aligned and of the same data length as the cache line. The cache line size is optimized for the system bus width, memory interleave, and CPU performance.

Because the bridge interfaces to the cache coherence domain of the system bus, it should issue transactions of the length and alignment of the cache line.

Continued on next page

Cache Line Size and Alignment

Cache line size and alignment will be different depending on the system design. Typical cache line sizes for Digital machines will be 64 bytes with a 64-byte alignment or 32 bytes and 32-byte alignment.

The I/O device works on systems with different line sizes, and is programmed for the cache line size through the compelled data length field. This allows the configuration process to optimize the I/O device for the particular system in which it is interfacing. The configuration process will assign a data length for the I/O module to use when transferring data to and from system memory. The I/O module should use the value programmed in the compelled data length field of the LOGICAL_MODULE_CONTROL CSR as the data length to use in the DL[2..0] field of the data and disconnection phase of compelled transfers. This value must also determine the address boundary over which a compelled transfer should not cross. This ensures that the data length and alignment of I/O transfers will be the same as the cache line size of the system.

Maintaining Transaction Alignment

If the node uses a data width other than the one enabled by the configuration process, the combination of the data length and the new data width will result in improper alignment with the system bus. Therefore, any node that uses a data width different from the one enabled should adjust the data length to maintain the transaction alignment programmed by the configuration process. For example, assume the configuration process enabled D64 and a data length of 0b011 (4 data beats). This implies 32-byte transactions. If the Futurebus+ node uses a data width of D32 for a memory transaction even though the configuration process enabled D64, the node should adjust the data length to 0b100 (8 data beats) to maintain the 32-byte alignment. Using a smaller data width will generally result in lower performance.

Continued on next page

Achieving High Performance

High performance is achieved by using the largest data width supported by the system, and by using addresses and transaction data lengths that align with those of the system bus and the cache line size. To obtain optimum performance through Futurebus+ bridges on Digital systems:

- Minimize the number of write partial transactions.
- Never use read partial transactions to memory.
- Minimize the number of connection/disconnection phases (without violating the data length and data width requirements).
- Use naturally aligned transactions of the data width and data length programmed by the configuration process as often as possible.

  Whenever possible, ensure that transactions align with the cache line addresses of the system and are of that size.
- Never attempt to cross the alignment boundary configured by the configuration process.

Byte-Aligned Addresses

Generally, the I/O modules are passed a buffer address, located in system memory, into which they will either read or write data. Since the I/O module cannot control the starting address, and therefore the address alignment, it must be able to transfer data to and from any byte-aligned address.

The Futurebus+ requires transactions to be bus-aligned to the data width of the transfer. That is, data sent using a D32 transaction will have the two least significant address bits assumed to be equal to zero by the bridge. Data sent using a D64 transaction will have the three least significant address bits assumed to be equal to zero by the bridge. This means that D32 transactions must be at least 4-byte aligned and D64 transactions must be at least 8-byte aligned.

I/O DMA Block Architecture

Overview

Devices that transfer sequential blocks of DMA data can increase performance by using the MORE (MR*) bit in the disconnection phase of split read and all write transactions. This bit gives the system bus bridge a *hint* that another sequential transaction will be forthcoming. Although the use of MR* is not required, it is recommended by Digital for increased performance.

Prefetching and Write Buffering

When memory latency through a bridge is long, it is desirable to be able to prefetch read data. When write transaction lengths are smaller than the memory bus write transaction lengths (normally a cache line), it is desirable to buffer the smaller write transactions into one larger write transaction to improve I/O and system performance. When data is written to or read from memory, it is more efficient to read or write an entire cache line. I/O traffic typically transfers data with a transaction length less than that of the system bus. It is beneficial to have the bus bridge buffer several writes into the proper cache line size, or to read the entire cache line and supply the data as it is requested by I/O device read transactions.

Continued on next page

Control Transaction

Read and write transaction ordering is relevant only to the consumer of data. During I/O transactions, a data buffer is turned over to the I/O module by the CPU. Once the buffer is owned by the I/O module, the CPU is not allowed to read or write the buffer until the transaction is complete and the host is notified in some DMA protocol-specific manner. This operation completion mechanism is referred to as a *control transaction*. This could consist of a completion interrupt or a write to a control data structure in memory. Until the control transaction is issued, the I/O module is allowed to read and write the buffer in any order.

For the bridge to perform the previously mentioned optimizations, it must be able to differentiate DMA block data transactions from control transactions.

Control transactions signal the end of the DMA transfer and must not be made visible to memory before all DMA write data is visible and prefetched DMA read data has been invalidated. Accordingly, the bridge must know that it can read ahead in the DMA buffer to prefetch valid data. The I/O module is the only device that knows this information. For prefetching and write buffering to work properly, there must be a way to communicate this information to the bridge.

Continued on next page

DMA Block-Copy Indicator

On the Futurebus+, a bit in the command field, CM1*, on the disconnection phase is used to indicate that a DMA block-copy operation is in progress. A DMA block-copy operation is defined to be the series of read or write transactions of the same data width and same data length, with each transaction in the block-copy having CM1* (labeled MR* in IEEE 896.1) asserted in the disconnection phase. The DMA block-copy is terminated by a transaction, from that transaction ID, with MR* released.

The bridge establishes a separate DMA block-copy based on the global ID of the master issuing the transactions. In this context, the global ID is defined to be the bus number (in the case where the bridge supports a hierarchy of bridges) and node ID concatenated with the transaction ID (AD[5..0] of the disconnection phase). This allows any node to have up to 64 concurrent DMA block-copy operations, each one independent of the other. All transactions in the same DMA block-copy must be issued with the same transaction ID.

Only read operations that have been split and write operations can supply the necessary disconnection data to initiate DMA block-copy operations. Connected read operations cannot take advantage of this feature.

Write Partial bus transactions cannot use the DMA block-copy feature.

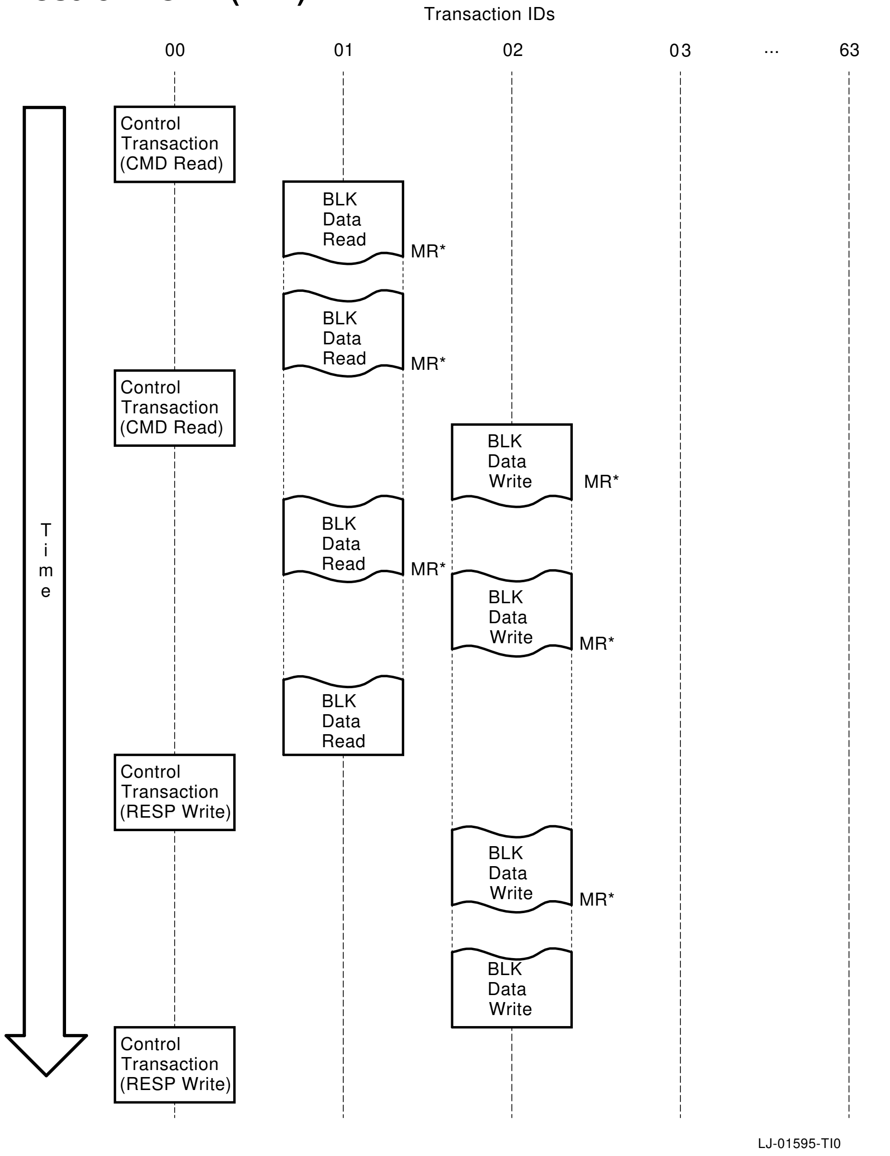
Continued on next page

Example of MORE (MR*)

Figure 2–12 provides an example of the use of MORE (MR*). In this case, assume the module is a typical I/O module that fetches commands from a command queue in memory and returns responses to a response queue, also in memory. In between, the I/O module also does some work.

Continued on next page

Figure 2–12 Use of MORE (MR*)



Continued on next page

Example of MORE (MR*) (continued)

The I/O module starts by reading a command from its command queue using a control transaction (for instance, a transaction that does not use MR*). In the example, this command is read using transaction ID #00 and directs the I/O module to begin a DMA block transfer from memory. The I/O module does so, using reads (that are split by the bridge), MR*, and transaction ID #01.

Eventually, the I/O module reads a second command from its command queue (again using a control transaction and transaction ID #00). This command directs the I/O module to begin a DMA block transfer to memory. The I/O module does so, using writes, MR*, and transaction ID #02.

The I/O module continues both DMA blocks until it completes the DMA block transfer from memory. The final DMA read transaction from memory *does not* assert MR*. The DMA read transaction does indicate transaction ID #01, which indicates to the system bus bridge that the DMA block represented by transaction ID #01 is now complete.

The I/O module then writes a response into its response queue using a control transaction and transaction ID #00.

The I/O module then resumes the DMA block transfer that it was performing to memory. The I/O module continues this DMA block until it completes it. The final DMA write transaction to memory *does not* assert MR* but does indicate transaction ID #02, which indicates to the system bus bridge that the DMA block represented by transaction ID #02 is now complete.

Continued on next page

Example of MORE (MR*) (continued)

Finally, the I/O module then writes a response into the response queue using a control transaction and transaction ID #00. This completes the example.

The example assumed that each of the two I/O buffers was entirely *physically* contiguous (which is possible for small I/O buffers). However, if either or both of the I/O buffers passed to the I/O module had spanned a discontiguous range of physical addresses, good design practice would have had the I/O module terminate the DMA block transfer each time it reached the point of an address discontinuity. It could then begin another DMA block transfer that spanned the next contiguous range of physical addresses.

Subset Implementations

Because bridges can choose not to implement prefetch buffers for all the possible potential transaction IDs, bridges that want to reduce the number of buffers required can allocate buffers based on a subset of the transaction IDs and some number of lower order ID bits. I/O modules should allocate transaction IDs starting with lower order transaction ID bits. If a node exceeds the number of IDs for which the bridge has storage, the bridge would not be able to prefetch and buffer for these particular IDs. Bridges could also choose to implement a limited number of prefetch buffers and allocate them on a first-come, first-served basis using a fully associative tag for the buffer address, and re-use these for other nodes as they become available.

Address Alignment

Addresses in a DMA block-copy operation must be naturally aligned to the transaction length. The only exception is the first transaction in a DMA read block-copy that might not be naturally aligned. Naturally aligned data is defined to be a data quantity of 2^n bytes whose least significant n address bits are zero. Thus, a naturally aligned D64 transaction with a data length of 4 beats must use connection phase addresses with the lower 5 address bits equal to 0.

Continued on next page

Extent of a DMA Block-Copy

The DMA block-copy operation is allowed to span multiple bus tenures by the I/O device. For the protocol to work for writes that are not split, the same disconnection data as specified for split reads must be supplied on the write disconnection phase. That is, the global identification and transaction ID must be supplied in the disconnection phase. A bridge can maintain separate read and write DMA block-copy buffers for any transaction ID.

Allowing DMA block-copy operations based on the transaction ID allows a node that contains more than one unit to have DMA block-copy operations for each unit, without having them interfere with each other. Also, a node can use a separate transaction ID to fetch or manipulate control structures, such as transaction queues or rings, without causing DMA block-copy operations in progress to terminate because of addresses being noncontiguous.

MR* and Virtual Addresses

Within Digital computer systems, DMA transfers are performed from data buffers that occupy a contiguous range of virtual addresses but may not occupy a contiguous range of physical addresses. (Each page in the virtually contiguous buffer may have its own unique virtual-to-physical address transformation.) Multiple bus transactions that are part of the same DMA block-copy operation must be made to physically contiguous addresses. The practical reality of this is that for Digital computer systems, DMA block-copy operations usually will not span multiple pages of the DMA buffer.

Nodes that do not support DMA block-copy must issue all transactions with disconnection field CM1* released. This ensures compatibility with nodes that do not support DMA block-copy operations.

Continued on next page

Restrictions Using MR*

Bridges in a system with a hierarchy of I/O buses should maintain the prefetch and write buffers for I/O devices attached to their local buses and not forward the MR* bit assertion onto other buses. This maintains any prefetched read data closer to the destination device and distributes the number of buffers.

Use of Transaction IDs

There are a total of 64 transaction IDs for each node. Nodes can use these IDs in any order.

Disconnection data, AD[5:0], represent transaction IDs for the master. This allows any node to have up to 64 outstanding transactions at a time.

The bridge maintains a DMA block-copy for each transaction ID; therefore, using multiple IDs for the same DMA block-copy becomes more complicated. There is no need to use multiple transaction IDs for writes since the ID can be reused once the transaction completes on the Futurebus+. Read transaction IDs cannot be reused until the read response for that ID is received. If a node wants to use multiple IDs for one DMA block-copy, the node should be certain to issue transactions such that the addresses of the transaction from different IDs are not within the prefetch quantity the bridge uses. Since this quantity varies depending on the system bridge, this quantity must be passed to the I/O module via the device initialization procedure.

Continued on next page

Read/Write Ordering

Reads to addresses within a DMA write block-copy are not guaranteed to return the data in the write block-copy.

DMA Read Block-Copy Ordering

For prefetching to work for reads, the I/O module guarantees that for any read transaction with CM1* asserted on the disconnection phase it can accept data that may already be contained in a prefetch buffer allocated for that transaction ID. The bridge must guarantee that any read transfer after the termination of the DMA block-copy will be serviced with a fresh copy of the data from memory. The first read transaction with CM1* asserted defines the start of the DMA block-copy for that transaction ID. Any transaction, following the start of a DMA read block-copy with CM1* released will terminate the DMA block-copy for that transaction ID.

On a read DMA block-copy, the transaction ID from the disconnection phase must be matched to one of the block-copy buffers the bridge is maintaining to ensure that a read from this node will not be serviced from the prefetch buffer of a different node. It is optional whether the bridge checks for proper addresses. Addresses in the DMA read block-copy should be monotonically ascending and must naturally align to the transaction size (except for the first read transaction of the block-copy). In addition, they should all be of the same data length and data width, up to the page or segment size of the system. When crossing a page or segment boundary, the address of the next transaction can be noncontiguous with the previous transaction address. It is assumed that any data prefetched by the bridge will not cross a page or segment boundary. A bridge can maintain as many DMA block-copy operations for which it has resources.

Continued on next page

DMA Read Block-Copy Termination

The terminating read transaction can be serviced from the prefetched data for that block-copy. Writes to addresses in a DMA read block-copy that have not yet been terminated are not guaranteed to result in a subsequent read transaction reading the most recently written data.

Response Identified by Transaction ID

Nodes must identify Read Response data using the transaction ID. The Read Response does not return the original address of the read request.

DMA Write Block-Copy Ordering

During write block-copy operations, the I/O module must allow any write transaction with CM1* set in the disconnection phase to be buffered into larger length memory transfers. The bridge guarantees that any write data with CM1* cleared on the disconnection phase is not made visible to memory before all other buffered data from that transaction ID is made visible to memory.

The bridge must check and match transaction IDs with the block-copy operations it is maintaining. It is optional whether the bridge checks for proper addresses in the block-copy. Transactions from other nodes and transactions from other IDs from the same node can bypass buffered writes from a DMA block-copy operation as long as the block-copy operation has not been terminated. Once a block-copy operation has been terminated, write ordering relative to this node ID must be preserved. The I/O module must properly terminate the block-copy operation before a termination interrupt is issued, to be sure the interrupt does not bypass any buffered writes. The terminating transaction of the block-copy operation cannot be the completion interrupt write transaction.

Continued on next page

DMA Write Block-Copy Ordering (continued)

Addresses in the DMA write block-copy must be monotonically ascending in naturally aligned transfers of all the same data width and data length, up to the page or segment size of the system. When crossing a page or segment boundary, the address of the next transaction can be noncontiguous with the previous transaction address. It is assumed that any data buffered by the bridge will not cross a page or segment boundary. As long as write transactions are part of a DMA block-copy, the bridge is allowed to send the write data to memory in any order.

DMA Write Block-Copy Termination

Any write transactions with CM1* released in the disconnection phase will terminate the DMA write block-copy operation for that transaction ID. The terminating write transaction can be buffered with previous writes in that block-copy if the address matches the next sequential block-copy address. A Write Partial transaction can be used to terminate a write block-copy.

Error Handling

In this context, aborting a DMA block-copy means invalidating any prefetch data or purging any buffered write data, without writing it to memory. Terminating a DMA block-copy means invalidating any prefetched read data and flushing any buffered write data to memory.

The bridge is not allowed to buffer and write a transaction that has a transaction error. The bridge must abort any DMA block-copy operations on a system reset.

I/O modules that do not retry transactions that encounter errors must initialize any DMA block-copy for that node as part of its error recovery mechanism.

Continued on next page

Error Handling (continued)

It is device-dependent whether a bridge aborts a DMA block-copy if addresses are not correctly ascending or if the transaction lengths differ once a block-copy is started. This should be considered an error condition and a violation of the protocol. At a minimum, the bridge should terminate the DMA block-copy and not supply the data from a prefetched read block-copy (in the case of reads), or not buffer the write data (in the case of a write block-copy).

Initializing a DMA Block-Copy

A means must be provided to instruct the bridge to invalidate any prefetched reads and to purge any buffered writes in the event of an error, or if the I/O module is initialized or fails. As part of the initialization and the BIST procedure, each I/O module that supports this prefetching protocol must clear any potential DMA block-copy in the bridge associated with its transaction IDs. The node must also initialize the DMA block-copy whenever it encounters an unrecoverable error, or is otherwise initialized.

A CSR location in the unit-specific address space of the bridge must be provided for this purpose. Writing the bus number, node number, and transaction IDs for I/O devices that are used for DMA block-copy operations to this register indicates that any DMA block-copy from this transaction ID should be invalidated. Prefetched read data must be invalidated and buffered write data must be discarded and not written to memory.

The initialize CSR is in the bridge unit-specific address space. This address must be communicated to the I/O module during the initialization process of the nodes. Because this operation might require the I/O node to issue a read or write to memory, the transaction ID used for this read must not be an ID used for any block-copy operations. The bridge could have buffered writes remaining from an aborted block-copy operation; therefore, a read or write to obtain the address of the initialize CSR for the bridge could cause a write to memory of the previously buffered block-copy. This would not be a desired consequence because the data is from a previous operation.

Futurebus+ Profile B CSRs

Overview

Table 2–2, Table 2–3, and Table 2–4, provide an overview of the Futurebus+ Profile B architectural CSRs. The following sections provide details of each individual CSR.

Byte Lane Mapping

Futurebus+ is designed to support both big endian and little endian architectures. Futurebus+ supports both endian byte lane mapping by fixing the byte lane address map such that all modules agree on where to find any byte offset on the same bus. This *address invariance* approach implies that all nodes see the same data structures or components of data structures at the same addresses.

For compatibility among architectures, the Core CSRs, Futurebus+ Dependent CSRs, and ROM CSRs are required by the IEEE standard to be in big endian data format.

NOTE

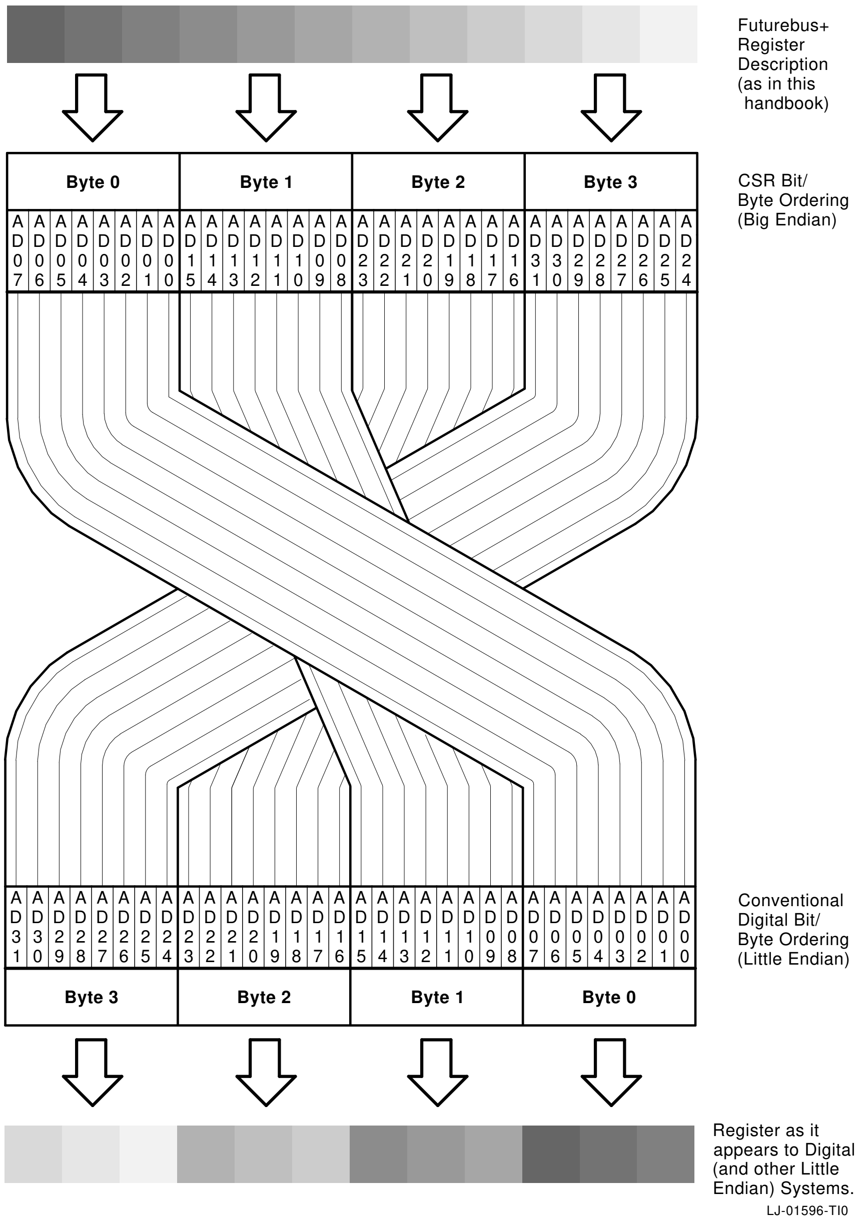
Most Digital systems are designed with a little endian architecture. Therefore, the system software (diagnostics, console, device drivers) will exchange the data format of the Core, Futurebus+, Dependent, and ROM CSRs to little endian format as needed.

Figure 2–13 illustrates this CSR mapping. The big endian data format is mapped onto the Futurebus+ Profile B bus. Big endian data formats will be exchanged by the software to a little endian data format on Digital systems. The bit positions are enumerated as the physical backplane signals in these figures.

Initial Unit Space CSRs, DMA, and DMA control structures can be either big or little endian mapped. For big endian CSRs and structures, the software on Digital systems must perform a little endian alignment for the CPU to interpret the data correctly.

Continued on next page

Figure 2–13 Byte Mapping of Little Endian Memory Data Into CSRs Via the Futurebus+



Futurebus+ Profile B CSR Overview

Control and Status Registers (CSRs)

Every Profile B node must have the minimum required CSRs listed in the following tables. Each of the CSRs is described briefly in these tables and described in more detail in the related section.

Table 2–2 Core CSRs

Register Name	Bit/Field Name	Mandate	Comment
NODE_ID	BUS_ADDRESS	Required	-
	GEOGRAPHICAL_ ADDRESS	Required	-
	NODE_SIDE	Required	-
RESET_START	-	Required	Bits reserved by IEEE 1212 are optional.
SPLIT_TIMEOUT_LO	-	Required in Masters	This is a 32-bit CSR with ≈233 ps resolution. Nodes shall implement at least a timeout range of 3.91 ms to 246.23 ms with a tolerance of ±10% of the value set.
TEST_START	CAT[0]	Required	CAT[0] = initialize category.
	CAT[1]	Recomm.	CAT[1] = extended
	TEST_STEP	Required	-
TEST_STATUS	TEST_STATE	Required	-

Continued on next page

Table 2–2 (Continued) Core CSRs

Register Name	Bit/Field Name	Mandate	Comment
ERROR_HI	ERROR_SUMMARY	Required	-
	MASTER	Required	-
	NONEXISTENT_ ADDR	Required in Masters	-
	COMMAND_PAR_ ERROR	Required	-
	DATA/ADDR_PAR_ ERROR	Required	-
	PROTOCOL_ERROR	Required	-
	TRANSACTION_ TIMEOUT	Required in Masters	-
	SPLIT_TIMEOUT	Required in Masters	-
	CAPABILITY_LINES	Required	Only CA2 required.
	STATUS_FIELD	Required	-
	COMMAND_FIELD	Required	-
ERROR_LO	BUSY_RETRY_ THRESHOLD_ EXCEEDED	Required in Masters	-
	LENGTH_ERROR	Required in packet mode only	-
	NONEXISTENT_ TRANSACTION_ID	Required	-

Continued on next page

Table 2–3 Futurebus+ Dependent CSRs

Register Name	Bit/Field Name	Mandate	Comment
LOGICAL_MODULE_ CONTROL	COMPELLED_DATA_ LENGTH	Required	-
	NON_CACHE_DATA_ WIDTH	Required	CSR_DW not required. Only data width of 32 bits allowed to CSRs for Profile B.
	64_BIT_ADDRESS	Required	-
	PARITY_REPORT_ ENABLE	Required	-
	MASTER_ENABLE	Required in Masters	-
BUS_PROPAGATION_ DELAY	-	Required	-
TRANSACTION_ TIMEOUT	-	Required	As defined in IEEE 896.1. This is a 32-bit CSR with ≈233 ps resolution. Nodes shall implement at least a timeout range of 30.5 μs to 457.7 μs (bits 17 to 20) with a tolerance of ±10% of the value set.
BUSY_RETRY_ COUNTER	Threshold	Required	16-bit CSR. Nodes shall implement the full 16-bit count for a threshold.

Continued on next page

Table 2–3 (Continued) Futurebus+ Dependent CSRs

Register Name	Bit/Field Name	Mandate	Comment
BUSY_RETRY_DELAY	Delay	Required	Nodes shall implement at least a delay range of 7.6 μs to 62.4 ms (bits 15 to 27) with a tolerance of ±10% of the value set.

Table 2–4 ROM CSRs

Register Name	Bit/Field Name	Mandate	Comment
Bus Identifier	-	Required	0x30383936 (ASCII "0896")
Profile ID	-	Required	Profile B ID is 0x42. All unused locations are zero.
MODULE_LOGICAL_ CAPABILITY	See footnote.[1]	Required	Profile B protocol requires that SPLIT_ACCEPTOR be implemented and its capability bit set if module is capable of becoming a master.
NODE_ CAPABILITIES_EXT	See footnote.[1]	Required	Profile B protocol requires the module to declare its data length capability. Nodes shall declare their data length capabilities. Modules shall support central arbiter.

[1] All bits describing the module's implemented capability shall be set to one. Nonimplemented features shall have the corresponding capability bits set to zero.

Continued on next page

Table 2–4 (Continued) ROM CSRs

Register Name	Bit/Field Name	Mandate	Comment
BUSY_RETRY_ CAPABILITY	2 CSRs	Required	-
MODULE_VENDOR_ ID	-	Required	-
MODULE_SPEC_ID	-	Required	-
MODULE_HW_ VERSION	-	Required	-
MODULE_SW_ VERSION	-	Recomm.	-
NODE_ CAPABILITIES	See footnote.[1]	Required	Profile B modules are required to support split timeout and BIST.

[1]All bits describing the module's implemented capability shall be set to one. Nonimplemented features shall have the corresponding capability bits set to zero.

Core CSRs

Definition

The Core CSRs are a set of registers that have a common definition across several IEEE bus standards (refer to the IEEE 1212 CSR standard).

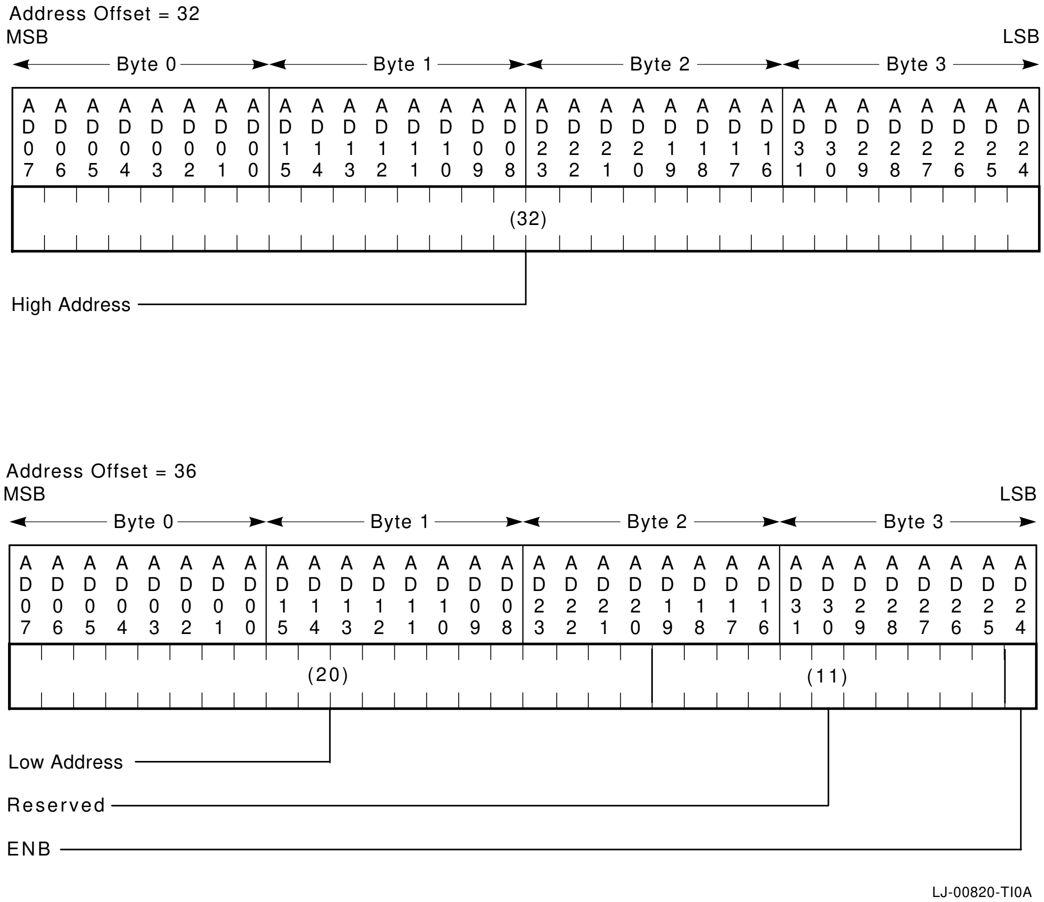
ARGUMENT CSRs

The format of ARGUMENT CSRs is shown in Figure 2–14.

NOTE

The results of writing the ARGUMENT CSRs during a test are unpredictable.

Figure 2–14 ARGUMENT CSRs Format



Continued on next page

Table 2–5 lists the ARGUMENT CSR bit definitions.

Table 2–5 ARGUMENT CSRs Bit Definitions

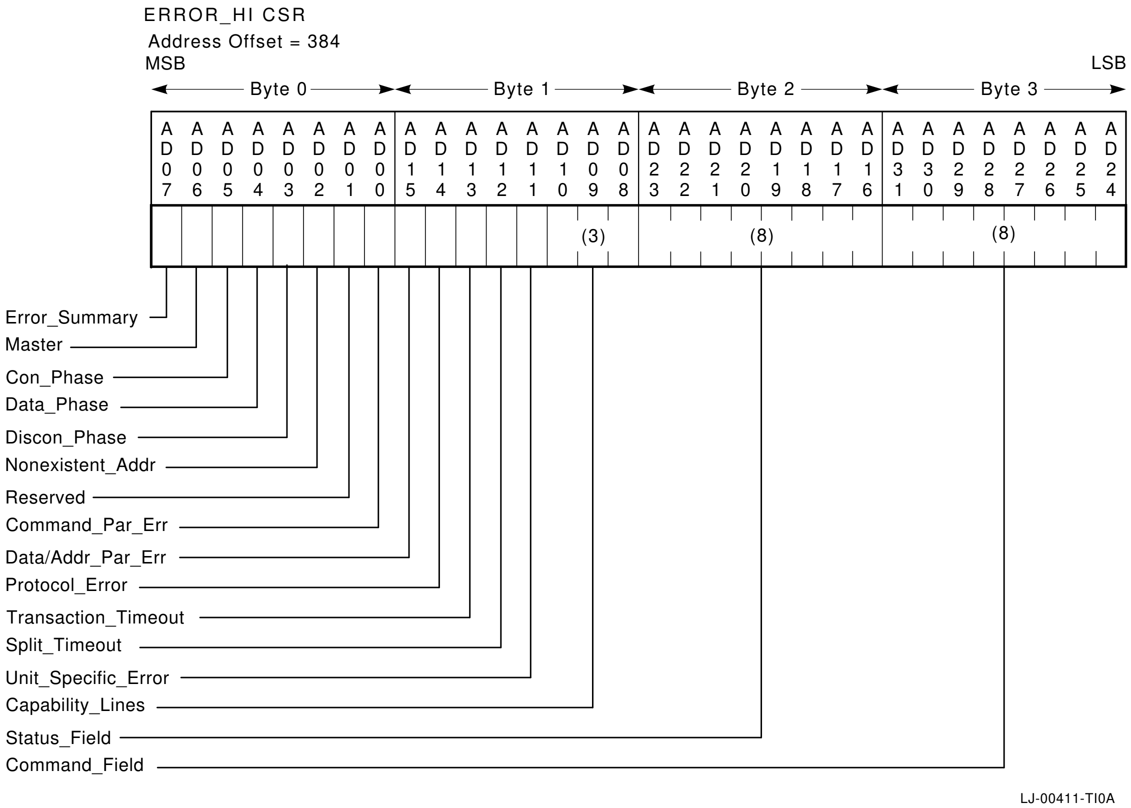
Field	Description	Access Type	Power Up	System Reset	Bus Initialize
High Address, Low Address	Contents specified when running external tests (CAT field in the TEST_START CSR = extended diagnostic). All other uses of this CSR are vendor-defined.	R/W	0	0	No change.
ENB	When this bit and the MASTER_ ENABLE bit in LOGICAL_ COMMON_CONTROL CSR are set, the node can perform tests that generate Futurebus+ transactions to remote (that is, off-board) memory, starting at the address in the ARGUMENT_HI /ARGUMENT_LO CSRs.	R/W	0	0	No change.

Continued on next page

ERROR_HI and ERROR_LO CSRs

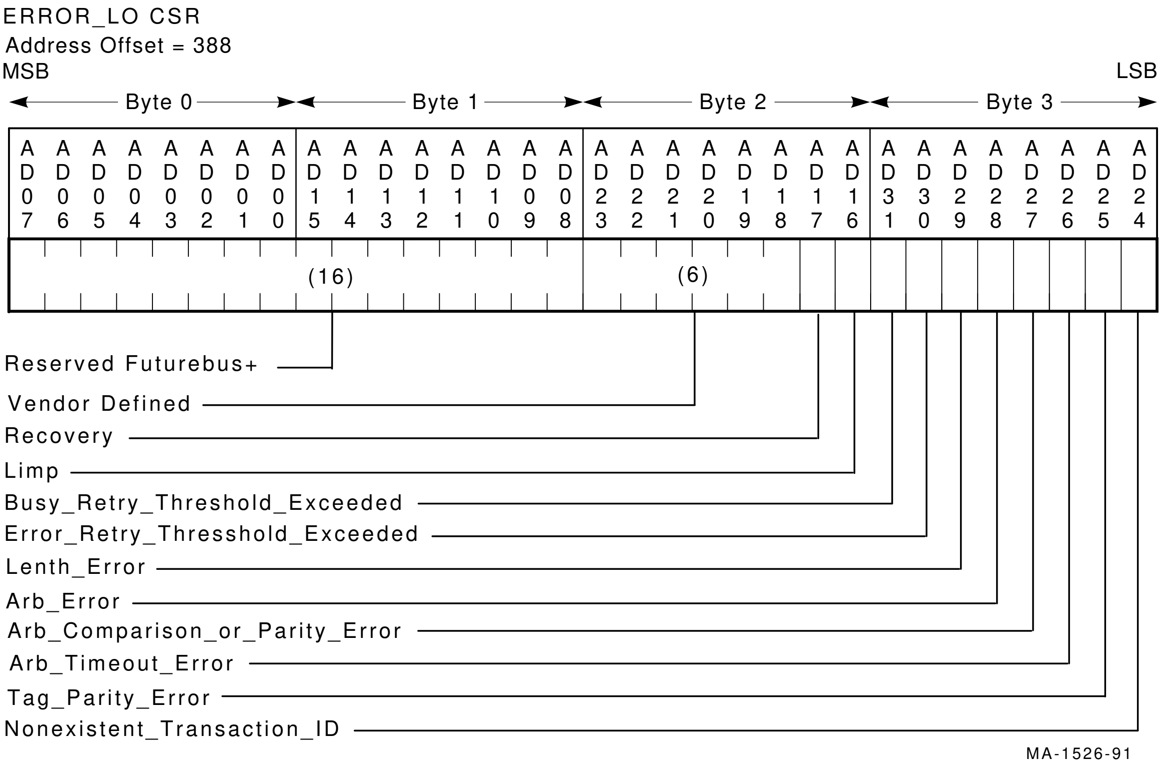
Figure 2–15 shows the ERROR_HI CSR format and Figure 2–16 shows the ERROR_LO CSR format.

Figure 2–15 ERROR_HI CSR Format



Continued on next page

Figure 2–16 ERROR_LO CSR Format



Continued on next page

Table 2–6 lists the ERROR_HI bit definitions and Table 2–7 lists the ERROR_LO CSR bit definitions.

Table 2–6 ERROR_HI CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
ERROR_ SUMMARY	This bit is the logical "or" of specified other bits in ERROR_HI and ERROR_LO. When this bit is set, the only other state bits in the error log CSRs that can change are RECOVERED and ERROR_RETRY_THRESHOLD_ EXCEEDED	RO	0	0	No change.
MASTER	This bit will be set to one if this node was master on the bus at the time the ERROR_SUMMARY bit was set.	RO	0	0	No change.
CON_PHASE	This bit is set if an error is detected during the connection phase.	RO	0	0	No change.
DATA_PHASE	This bit is set if an error is detected during the data phase.	RO	0	0	No change.
DISCON_ PHASE	This bit is set if an error is detected during the disconnection phase.	RO	0	0	No change.
NONEXISTENT_ ADDR	This bit is set by a master when no slave acknowledges an address (no SL* during the connection phase). If set, this bit results in the setting of the ERROR_ SUMMARY bit.	R/W1C	0	0	No change.
Reserved	Reserved by IEEE 1212.	-	-	-	-

Continued on next page

Table 2–6 (Continued) ERROR_HI CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
COMMAND_ PAR_ERROR	This bit is set by any module during the connection phase and by any participating slave in other phases when a parity error is detected on the command bus. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.
DATA/ADDR_ PAR_ERROR	This bit is set by any module during the connection phase or by the master or any participating slave during the data and disconnection phases when a parity error is detected on the data/address or data bus. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.
PROTOCOL_ ERROR	This bit is set by a master or slave during the connection phase when an unsupported command or capability or an illegal operation is decoded. If set, this bit results in the setting of the ERROR_ SUMMARY bit.	R/W1C	0	0	No change.
TRANSACTION_ TIMEOUT	This bit is set when a transaction timeout is detected during any phase (IEEE 896.1, Chapter 6). If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.
SPLIT_ TIMEOUT	This bit is set by the original requester when the split transaction timer expired before a response returned. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.

Continued on next page

Table 2–6 (Continued) ERROR_HI CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
UNIT_ SPECIFIC_ ERROR	This bit is set by a master or participating slave when an error that is not visible on the bus causes data corruption. If any bit in a unit specific error CSR is set, then this bit will be set. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.
CAPABILITY_ LINES	This field is locked when an error is detected. It is loaded by a participating slave with the state driven on the capability lines at the time the first error is detected. It is loaded by a master during the connection phase and locked when an error is signaled on the status bus or is detected by the master.	RO	0	0	No change.
STATUS_FIELD	This field is locked when an error is detected. It is loaded with the status driven on the status bus at the time the first error is detected. If this field is locked, the ERROR_ SUMMARY bit will be set.	R/W1C	0	0	No change.
COMMAND_ FIELD	This bit is loaded by a participating slave with the command driven on the command bus at the time the first error is detected. It is loaded by a master during the connection phase and locked when an error is signaled on the status bus or is detected by the master.	RO	0	0	No change.

Continued on next page

Table 2–7 ERROR_LO CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
Reserved Futurebus+	Reserved by IEEE 1212.	-	-	-	-
VENDOR_ DEPENDENT	This field can contain VENDOR_ DEPENDENT error bits.	-	-	-	-
RECOVERED	This bit is set when a transient fault occurred but later disappeared. This bit is set if there is a successful error retry, and not set if the LIMP bit is set.	R/W1C	0	0	No change.
LIMP	This bit is one if the node remains operational after a permanent fault is detected. This bit is implemented if the node remains operational in the presence of a permanent fault.	R/W1C	0	0	No change.
BUSY_RETRY_ THRESHOLD_ EXCEEDED	The number of retries that exceeded the threshold in BUSY_ RETRY_COUNTER CSR. No further retries of this transaction will be attempted. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.
ERROR_RETRY_ THRESHOLD_ EXCEEDED	The number of retries that exceeded the threshold in BUSY_ RETRY_COUNTER CSR. No further retries of this transaction will be attempted. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.

Continued on next page

Table 2–7 (Continued) ERROR_LO CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
LENGTH_ ERROR	This bit is set in data mode when more or less data than expected was received. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.
ARB_ERROR	This bit is set by the current master and all competitors during distributed arbitration or arbitration bus Phase 3 if both AC0* and AC1* are asserted. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.
ARB_ COMPARISON_ OR_PARITY_ ERROR	Set by any competitor during distributed arbitration or arbitration bus Phase 3 if parity checking is disabled and the value driven on AB[ ]* is less than the competitor's own competition number. When parity checking is enabled, this bit is set by any participating slave when a parity error is detected during Phase 3 or arbitration. If set, this bit results in the setting of the ERROR_ SUMMARY bit.	R/W1C	0	0	No change.
ARB_TIMEOUT_ ERROR	This bit is set in distributed arbitration or arbitration bus Phase 2 or 4 by any competitor when no competitor asserted AQ* (WIN), as described in the Arbitration (Distributed Arbitration) Bus Lines section of Chapter 3 in this handbook. If set, this bit results in the setting of the ERROR_SUMMARY bit.	R/W1C	0	0	No change.

Continued on next page

Table 2–7 (Continued) ERROR_LO CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
TAG_PARITY_ ERROR	This bit is set by any master or participating slave when a parity error is detected on the TAG bus. If set, this bit results in the setting of the ERROR_SUMARY bit.	R/W1C	0	0	No change.
NONEXISTENT_ TRANSACTION_ ID	This bit is set by the slave in a split response transaction when the transaction ID returned does not match that of any outstanding transaction. If set, this bit results in the setting of the ERROR_ SUMMARY bit.	R/W1C	0	0	No change.

Continued on next page

NODE_ID CSR

The NODE_ID CSR format is shown in Figure 2–17.

Figure 2–17 NODE_ID CSR Format

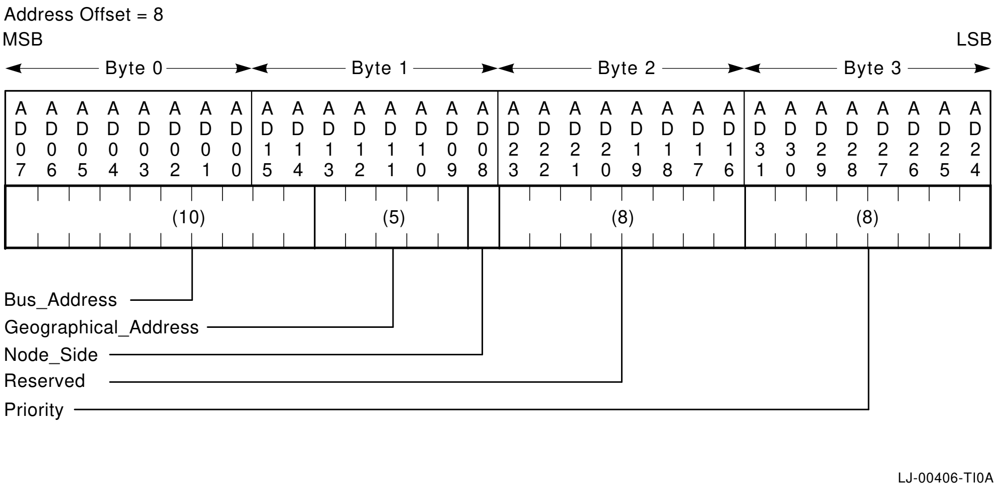


Table 2–8 lists the NODE_ID CSR bit definitions.

Table 2–8 NODE_ID CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
BUS_ADDRESS	This bit sets the BUS_ADDRESS field to the current bus address. It is not expected that this field will change while there are outstanding split transactions.	R/W	1023	1023	No change.

Continued on next page

Table 2–8 (Continued) NODE_ID CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
GEOGRAPHICAL_ ADDRESS	This format reflects the geographical address signals applied to the GA[4:0]* lines at reset time and is not modified by software, or any form of reset. All nodes in a module have identical values in this field.	RO	No change.	No change.	No change.
NODE_SIDE	This format is implicit in the module design and is not modified by software, any form or reset, or board placement. If the module contains two Futurebus+ nodes, the manufacturer assigns different values to their NODE_SIDE fields. If the module contains one node, the manufacturer assigns the value of zero.	RO	No change.	No change.	No change.
Reserved	Reserved by IEEE 1212.	-	-	-	-
PRIORITY	This field determines the 8-bit encoding to use for setting the priority field, PR[7:0] (defined in IEEE 896.1, Chapter 5), during distributed arbitration or arbitration bus when no other priority value is provided to the node. Reads of this field will return the last written value.	R/W	0	0	No change.

Continued on next page

RESET_START CSR

The RESET_START CSR format is shown in Figure 2–18.

NOTE

Writing this CSR causes a node reset. A node reset initializes the state of the node and starts the initialization tests.

The effect of a system reset and a write to the RESET_ START CSR shall be the same. The Futurebus+ interface logic shall not become unaligned to the bus.

Figure 2–18 RESET_START CSR Format

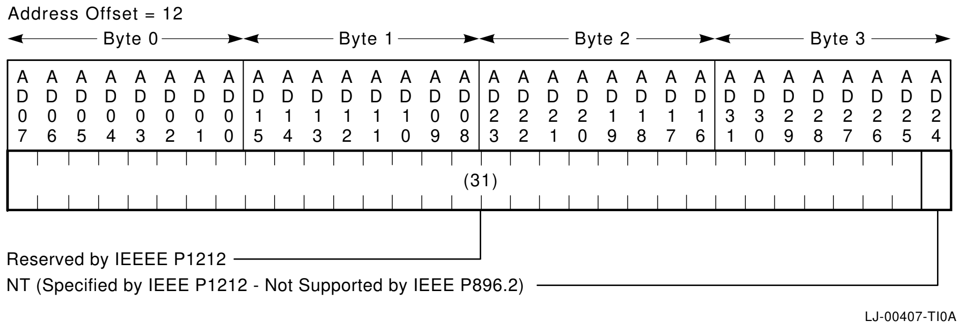


Table 2–9 lists the RESET_START CSR bit definitions.

Table 2–9 RESET_START CSR Bit Definitions

Field	Description	Access Type
Reserved	Reserved by IEEE 1212	-
NT	(Specified by IEEE 1212, and not supported by IEEE P896.2.)	-

Continued on next page

SPLIT_TIMEOUT (SPLIT_TO) CSR

The SPLIT_TIMEOUT CSR format is shown in Figure 2–19 and Figure 2–20. It is required if the node supports split transactions. This CSR is mechanized as a pair of independent 32-bit CSRs.

Figure 2–19 SPLIT_TIMEOUT CSR Format (a)

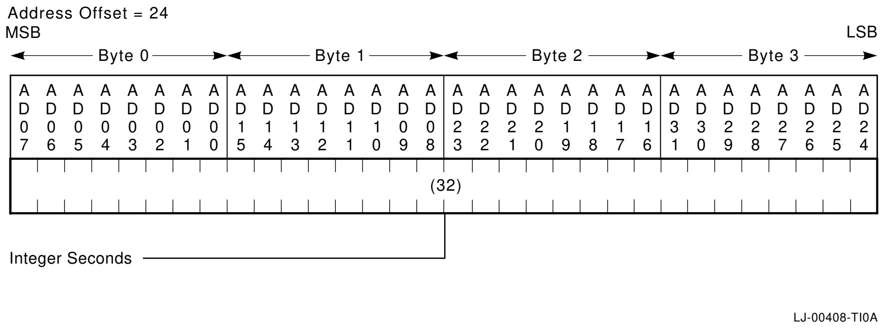
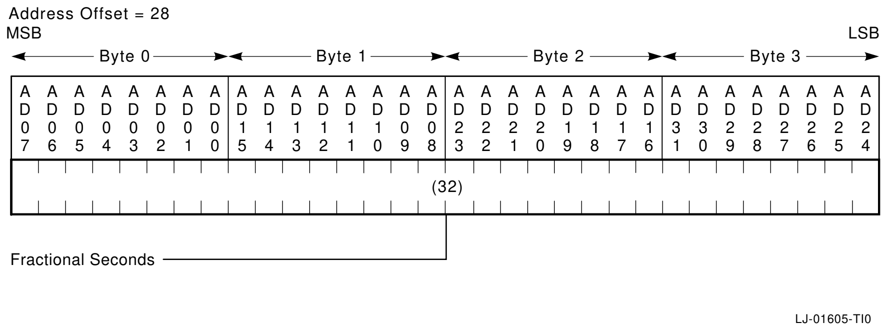


Figure 2–20 SPLIT_TIMEOUT CSR Format (b)



Continued on next page

Table 2–10 lists the SPLIT_TIMEOUT CSR bit definitions.

Table 2–10 SPLIT_TIMEOUT (SPLIT_TO) CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
Integer Seconds (HI)	Integer Seconds, address offset 24. This is required if the node supports split transactions.	R/W	0	0	No change.
Fractional Seconds (LO)	Fractional Seconds (in units of 2^{-32} seconds, $\approx$233 ps), address offset 28. This is required if the node supports split transactions.	R/W	0	0	No change.

Continued on next page

TEST_START CSR

Writing to this CSR shall cause the diagnostic test specified by the TEST_STEP field to begin execution. Following a system reset, this CSR shall be set to all zeros.

NOTE

The results of writing the TEST_START CSR during a test are unpredictable. The exception to this is the CAT field, which may be written with 0b0000 to halt testing.

The TEST_START CSR format is shown in Figure 2–21.

Figure 2–21 TEST_START CSR Format

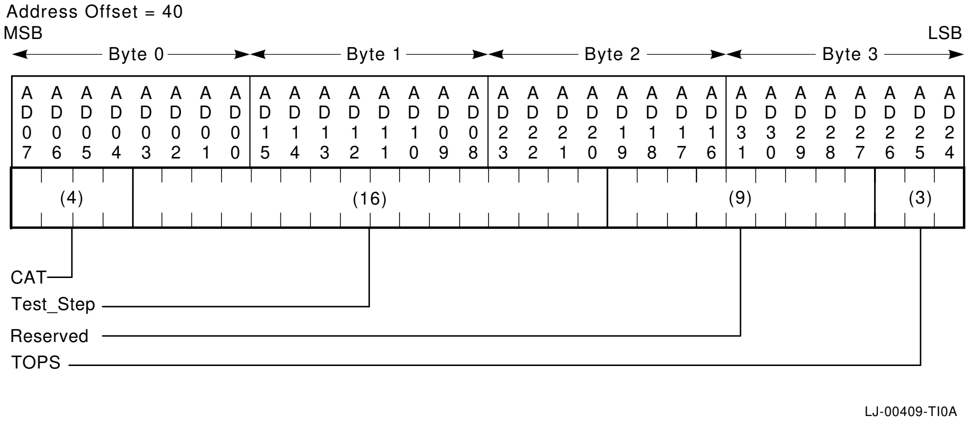


Table 2–11 shows the TEST_START CSR bit definitions.

Continued on next page

Table 2–11 TEST_START CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
CAT	This 4-bit field specifies the category of tests. Any combination of the bits may be valid. Destructive testing is only permitted in the "manual" category (for example, formatting a disk, or running a read/write exerciser.) See Table 2–12 for details.	R/W	0	0	No change.
TEST_STEP	This 16-bit field specifies the identifier of the test or test sequence to be run. The location and size of this field are standardized to simplify the execution of tests from ROM-based parameters. The interpretation of this field (identification of the test or test sequence) is vendor-dependent. TEST_STEP equal to 0x0000 indicates the default test(s) (maximum run time of default tests to not exceed 10 seconds). Default initialization category tests are the set of tests run on System Reset. TEST_STEP equal to 0xFFFF indicates that all tests in the category will be run.	R/W	0	0	No change.
Reserved	Reserved	-	0	0	0
TOPS	This 3-bit field (test options) specifies how the test is to be run. More than one bit may be set at a time (see Table 2–13).	R/W	0	0	No change.

Continued on next page

Table 2–12 Category

Value Binary	Name	Description
0b0001	CAT[0]	Initialization–Tests that shall not require any external resources. Every test that can be run without support or cooperation with other nodes or resources in the system shall be included in the initialization category.
0b0010	CAT[1]	Extended–Tests that shall use a buffer external to the unit under test as scratch space. The handle or pointer to the scratch space is passed to tests in the extended category in the ARGUMENT CSRs. To be consistent with the CSR standard, the buffer shall be 4 kbytes in size.
0b0100	CAT[2]	System–Tests that may require additional system resources beyond those required by a test in the extended category. The exact use of this category is defined by the vendor.
0b1000	CAT[3]	Manual intervention–All tests that require operator intervention or may be destructive shall only be included in this category.
0b0000	CAT[3..0] = 0	HALT–The all-0s encoding of the CAT[ ] field is reserved in the standard to force the current test sequence to stop. This encoding shall cause any test or test sequence that is currently active to halt after the current test completes. This is a graceful abort.

Continued on next page

Table 2–13 Test Options

Field Decimal	Value Binary	Name	Description
0	0b000	Halt on Error	Halt test if an error is detected.
1	0b001	Continue	Continue test sequence if an error is detected.
2	0b010	Loop on Error	Loop on the failing test if an error is detected; otherwise, halt when done.
3	0b011	Loop on Error	Same as 2.
4	0b100	Loop	Continuously loop until halted.
5	0b101	Loop	Same as 4.
6	0b110	Loop and Loop on Error	Continuously loop until halted. If an error is detected, loop on the failing test.
7	0b111	Loop and Loop on Error	Same as 6.

Continued on next page

TEST_STATUS CSR

The TEST_STATUS CSR format is shown in Figure 2–22.

Figure 2–22 TEST_STATUS CSR Format

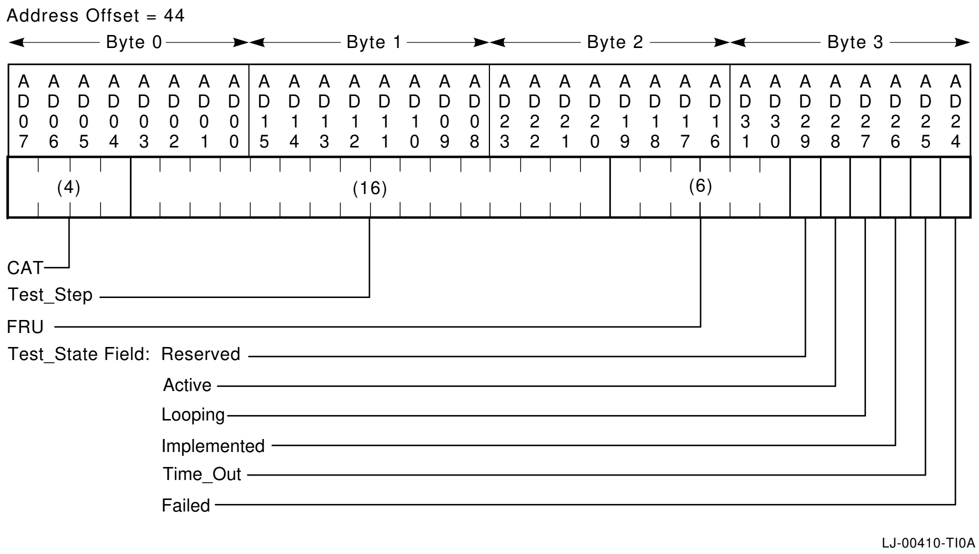


Table 2–14 shows the TEST_STATUS CSR bit definitions.

Continued on next page

Table 2–14 TEST_STATUS CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
CAT	This 4-bit field indicates the category of test that passed/failed when Halted is set in the TEST_STATE field. The contents of the CAT field are the same categories as used in the TEST_START CSR and shown in Table 2–12.	RO	0	0	no change
TEST_STEP	Depending on the contents of TEST_STATE, this 16-bit field specifies the identifier of the test step currently running the test step in which an error was detected, or the most recently completed test step. In the system and manual intervention categories, the interpretation of TEST_STEP is vendor-dependent.	RO	0	0	No change.
FRU	This 6-bit field specifies the identify of the failed component (vendor-dependent).	RO	0	0	No change.
TEST_STATE	This 6-bit field describes the results of the current test. More than 1 bit may be set at a time (see Table 2–15).	RO	0	0	No change.

Continued on next page

Table 2–15 Test Results

Bits Within the Field

Value	Name	Description
0x20	Reserved	Reserved
0x10	Active	Test active
0x08	Looping	Test looping
0x04	Implemented	Test implemented
0x02	Timeout	Timeout error detected
0x01	Failed	Fault detected

Interpretation of Values

Value	Name	Description
0x00	Successful	No test implemented
0x01	Illegal	(Test failed, not implemented)
0x02	Illegal	(Test failed, not implemented)
0x03	Illegal	(Test failed, not implemented)
0x04	Successful	Test complete, no error detected
0x05	Failed	Fault detected
0x06	Timeout	Timeout detected
0x07	Failed, Timeout	Timeout, fault detected
0x08	Illegal	(Test looping, not active)
0x09	Illegal	(Test looping, not active)
0x0A	Illegal	(Test looping, not active)
0x0B	Illegal	(Test looping, not active)
0x0C	Illegal	(Test looping, not active)
0x0D	Illegal	(Test looping, not active)

Continued on next page

Table 2–15 (Continued) Test Results

Interpretation of Values

Value	Name	Description
0x0E	Illegal	(Test looping, not active)
0x0F	Illegal	(Test looping, not active)
0x10	Checking	Test active, verification test implemented
0x11	Illegal	(Test active, fault detected, test not implemented)
0x12	Illegal	(Test active, fault detected, test not implemented)
0x13	Illegal	(Test active, fault detected, test not implemented)
0x14	Active	Test active (and implemented)
0x15	Failed	Test active, fault detected, test continued
0x16	Timeout	Test active, timeout error detected, test continued
0x17	Failed, Timeout	Test active; fault, timeout error detected, test continued
0x18	Illegal	(Test active, not implemented)
0x19	Illegal	(Test active, not implemented)
0x1A	Illegal	(Test active, not implemented)
0x1C	Looping	Test active, looping
0x1D	Looping	Test active, looping, fault detected
0x1E	Looping	Test active, looping, timeout
0x1F	Looping	Test active, looping, timeout, fault detected

Futurebus+ Dependent CSRs

Definition

The Futurebus+ Dependent CSRs are IEEE standard CSRs that are specific to the IEEE Futurebus+ 896 standard.

BUS_PROPAGATION_DELAY CSR

The BUS_PROPAGATION_DELAY CSR format is shown in Figure 2–23.

Figure 2–23 BUS_PROPAGATION_DELAY CSR Format

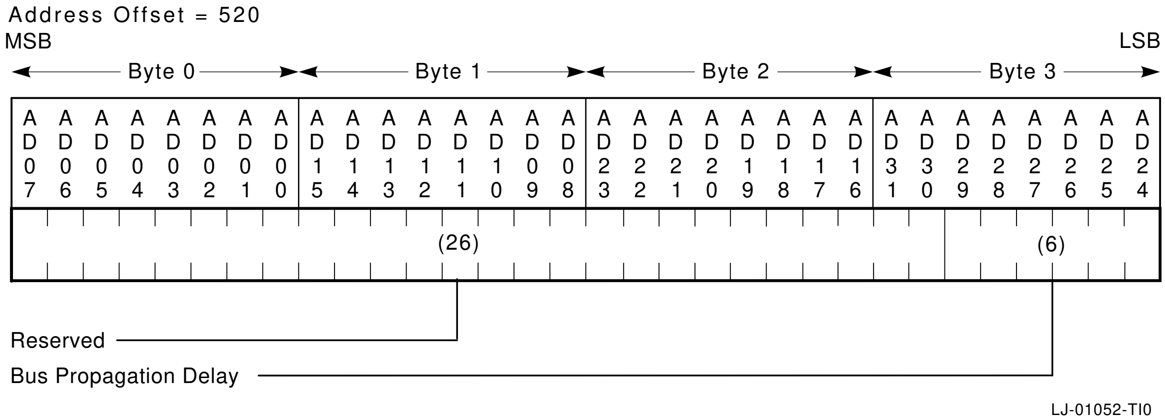


Table 2–16 lists the BUS_PROPAGATION_DELAY CSR bit definitions.

Continued on next page

Table 2–16 BUS_PROPAGATION_DELAY CSR Bit Definitions

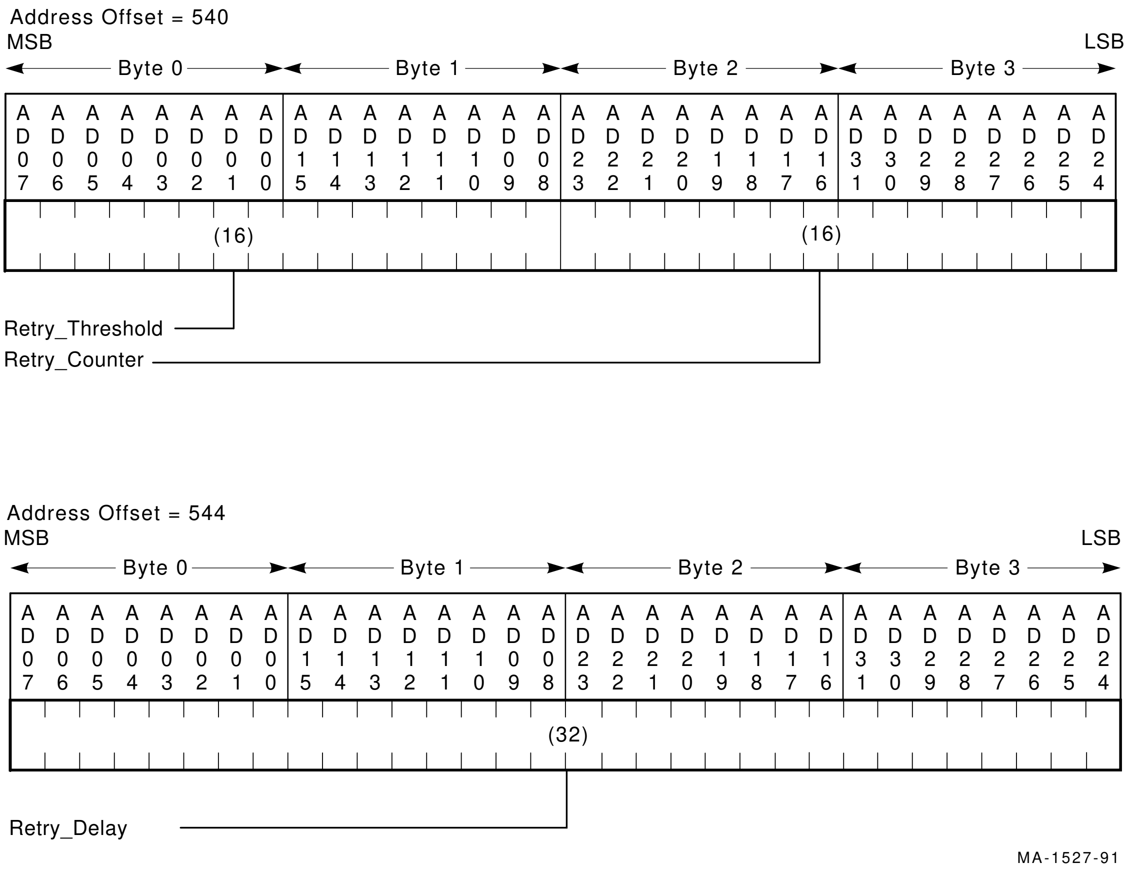
Field	Description	Access Type	Power Up	System Reset	Bus Initialize
BUS_ PROPAGATION_ DELAY	Following a system reset or bus initialize, this CSR will be set to 0x3F in the least significant 6 bits (≈14.7 ns). The most significant 26 bits of this CSR are reserved. The field will be in steps of 2^{-32} seconds (≈233 ps). The delay value is for one-way transit on the bus in accordance with IEEE 896.1. All nodes in a module will have identical values in their BUS_PROPAGATION_DELAY CSRs.	R/W	0x3F	0x3F	0x3F

Continued on next page

BUSY_RETRY CSRs

The BUSY_RETRY_COUNTER and BUSY_RETRY_DELAY CSR formats are shown in Figure 2–24.

Figure 2–24 BUSY_RETRY_COUNTER and BUSY_RETRY DELAY CSR Formats



Continued on next page

Table 2–17 lists the BUSY_RETRY_COUNTER CSR bit definitions.

Table 2–18 lists the BUSY_RETRY_DELAY CSR bit definitions.

Table 2–17 BUSY_RETRY_COUNTER CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
RETRY_ THRESHOLD	This bit field is expected to be set by the system configuration process based on values it extracts from the BUSY_RETRY_ COUNTER_CAPABILITY CSRs of all system nodes. If this field contains zero, busy retries will not be attempted.	R/W	0	0	No change.
RETRY_ COUNTER	This bit field specifies the number of retries to be attempted for each busy resource. This field will be set to zero upon completion of each transaction.	R/W	0	0	No change.

Table 2–18 BUSY_RETRY_DELAY CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
RETRY_ DELAY	This time value is in units of 2^{-32} seconds ($\approx$233 ps). The value can be changed to enable, for example, the retry time to be exponentially increased.	R/W	0	0	No change.

Continued on next page

Futurebus+ Dependent CSRs, Continued

ERROR_RETRY CSRs

The ERROR_RETRY_COUNTER and ERROR_RETRY_DELAY CSR formats are shown in Figure 2–25 and Figure 2–26.

Figure 2–25 ERROR_RETRY_COUNTER CSR Format

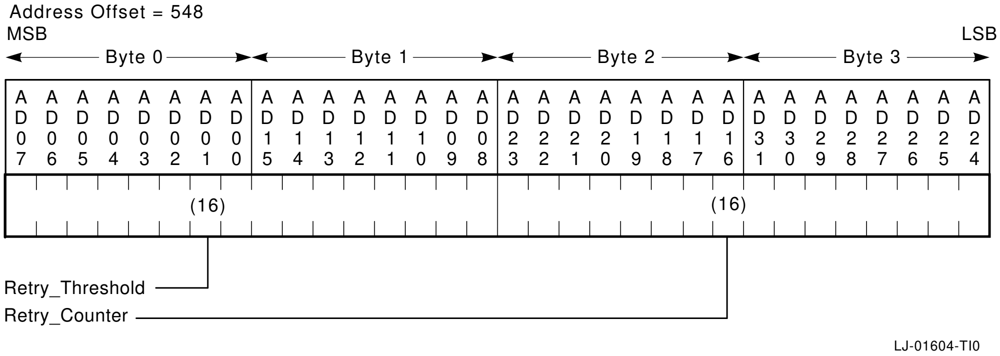
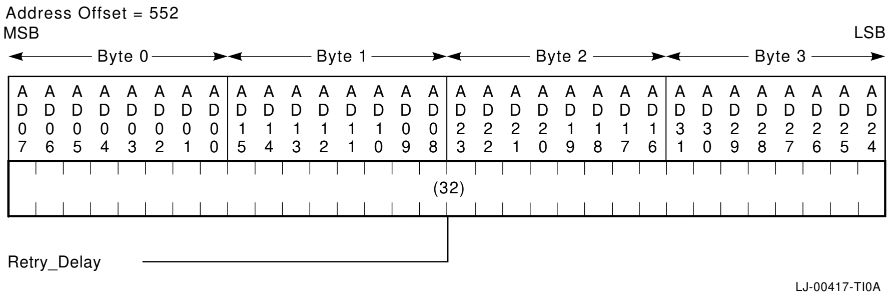


Figure 2–26 ERROR_RETRY_DELAY CSR Format



Continued on next page

Table 2–19 lists the ERROR_RETRY_COUNTER CSR bit definitions and Table 2–20 lists the ERROR_RETRY_DELAY CSR bit definitions.

Table 2–19 ERROR_RETRY_COUNTER CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
RETRY_ THRESHOLD	This bit field is expected to be set by the system configuration process based on values it extracts from the ERROR_RETRY_ COUNTER_CAPABILITY CSRs of all system nodes. If this field contains zero, error retries will not be attempted.	R/W	0	0	No change.
RETRY_ COUNTER	This bit field specifies the number of retries to be attempted for each error at the target resource. This field will be set to zero upon completion of each transaction.	R/W	0	0	No change.

Table 2–20 ERROR_RETRY_DELAY CSR Bit Definitions

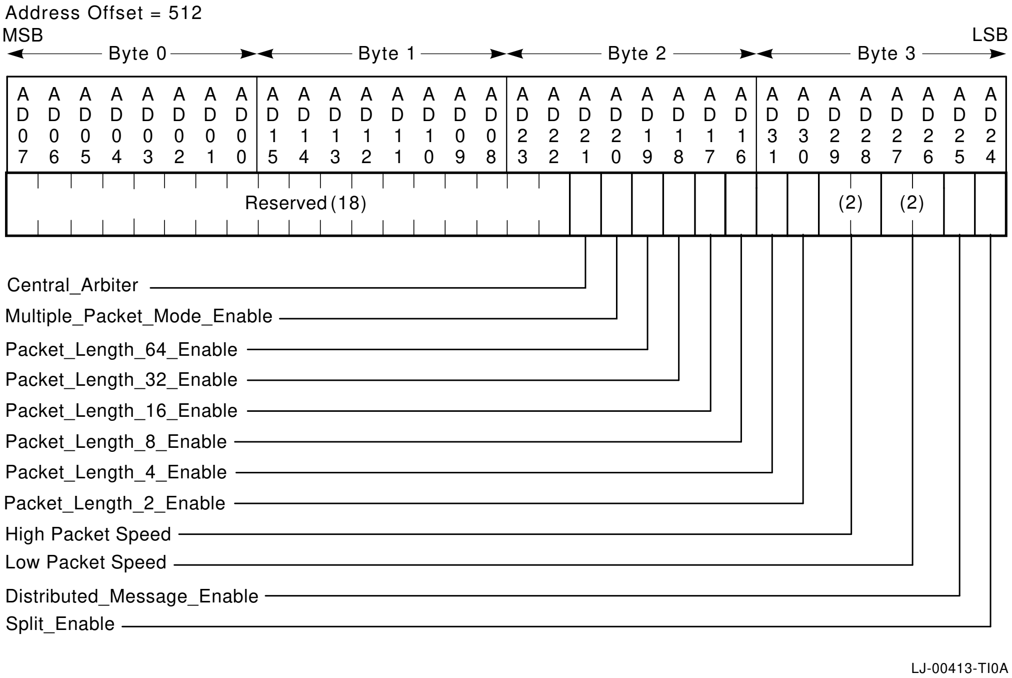
Field	Description	Access Type	Power Up	System Reset	Bus Initialize
RETRY_ DELAY	This time value is in units of 2^{-32} seconds ($\approx$233 ps). The value can be changed to enable, for example, the retry time to be exponentially increased.	R/W	0	0	No change.

Continued on next page

LOGICAL_COMMON_CONTROL CSR

The LOGICAL_COMMON_CONTROL CSR format is shown in Figure 2–27.

Figure 2–27 LOGICAL_COMMON_CONTROL CSR Format



Continued on next page

Table 2–21 lists the LOGICAL_COMMON_CONTROL CSR bit definitions.

Table 2–21 LOGICAL_COMMON_CONTROL CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
Reserved					
CENTRAL_ ARBITER	The system configuration process sets this bit to select the central arbiter instead of the distributed arbiter to allocate bus mastership. This bit is also set by hardware when REf transitions to logic zero following a powerup or system reset if a central arbiter is present in the backplane. This bit is expected to be set to the same state on all master-capable nodes in a system.	R/W	1 (central arb present)	1 (central arb present)	No change.
MULTIPLE_ PACKET_ MODE	If one, this bit specifies that the node has multiple transfer mode enabled.	R/W	0	0	0
PACKET_ LENGTH_n_ ENABLE	The system configuration process sets these bits to indicate which lengths the module is allowed to use. This 6-bit field contains a bit for each allowable size n, where n can be 64, 32, 16, 8, 4, or 2.	R/W	0	0	0
HIGHT_ PACKET_ SPEED	The system configuration process sets this field to one of four values to indicate the speed the module can use during high speed transfers.	R/W	0	0	0

Continued on next page

Table 2–21 (Continued) LOGICAL_COMMON_CONTROL CSR Bit Definitions

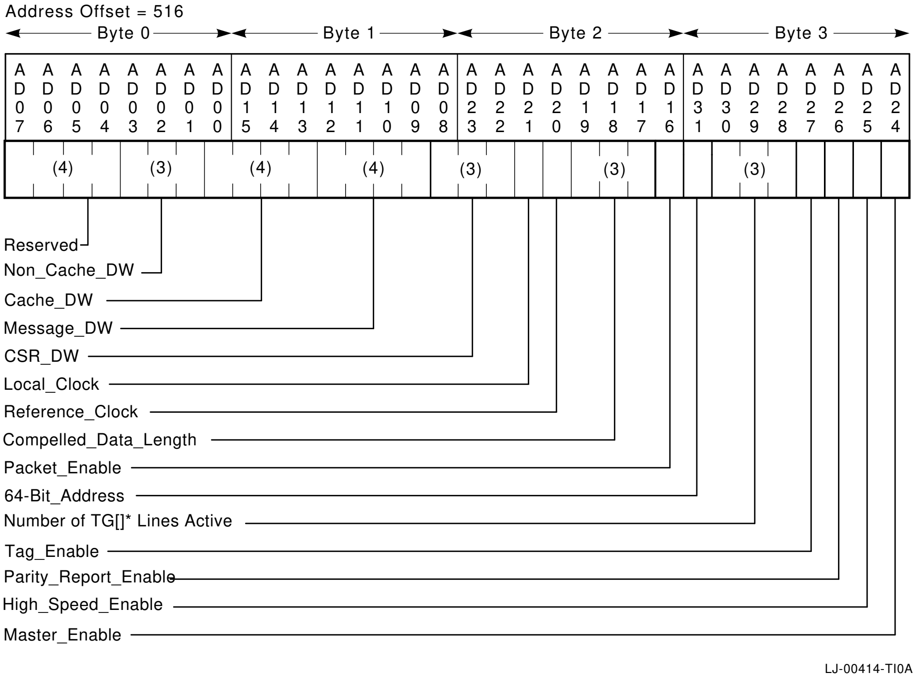
Field	Description	Access Type	Power Up	System Reset	Bus Initialize
LOW_ PACKET_ SPEED	The system configuration process sets this field to one of four values to indicate the speed the module can use during low speed transfers.	R/W	0	0	0
DISTRIBUTED_ MESSAGE_ ENABLE	The system configuration process sets this bit to allow the module to send distributed arbitration messages.	R/W	0	0	0
SPLIT_ ENABLE	The system configuration process sets this bit to indicate that the module can split bus transactions if it has the capability.	R/W	0	0	0

Continued on next page

LOGICAL_MODULE_CONTROL CSR

The LOGICAL_MODULE_CONTROL CSR format is shown in Figure 2–28.

Figure 2–28 LOGICAL_MODULE_CONTROL CSR Format



Continued on next page

Table 2–22 lists the LOGICAL_MODULE_CONTROL CSR bit definitions.

Table 2–22 LOGICAL_MODULE_CONTROL CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
Reserved	-	-	-	-	-
NON_CACHE_DW	This 3-bit field specifies the permitted noncached data path widths. A data path width of 32 bits is always supported. Split response data width will be the same data width as that of the original request.	R/W	0	0	No change.
CACHE_DW	This 4-bit field specifies the permitted data widths of cache transfers.	R/W	0	0	No change.
MESSAGE_DW	This 4-bit field specifies the permitted data widths of message transfers.	R/W	0	0	No change.
CSR_DW	This 3-bit field specifies the permitted data widths of CSR access. A data transfer width of 32 bits is always permitted. Split response data width will be the same data width as that of the original request.	R/W	0	0	No change.
LOCAL_CLOCK	If one, this bit specifies that this node contains a clock that is subordinate to a system reference clock.	R/W	0	0	0
REFERENCE_CLOCK	If one, this bit specifies that this node contains the system reference clock.	R/W	0	0	0

Continued on next page

Table 2–22 (Continued) LOGICAL_MODULE_CONTROL CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
COMPELLED_ DATA_LENGTH	This field enables the specific maximum compelled data length selected from those available as specified in the NODE_ CAPABILITIES_EXT CSR. The ED* signal preempts this field.	R/W	0	0	0

Field Value Decimal	Binary	Description
0	0b000	Unrestricted
1	0b001	Use compelled data length equal 1
2	0b010	Use compelled data length equal 2
3	0b011	Use compelled data length equal 4
4	0b100	Use compelled data length equal 8
5	0b101	Use compelled data length equal 16
6	0b110	Use compelled data length equal 32
7	0b111	Use compelled data length equal 64

Continued on next page

Table 2–22 (Continued) LOGICAL_MODULE_CONTROL CSR Bit Definitions

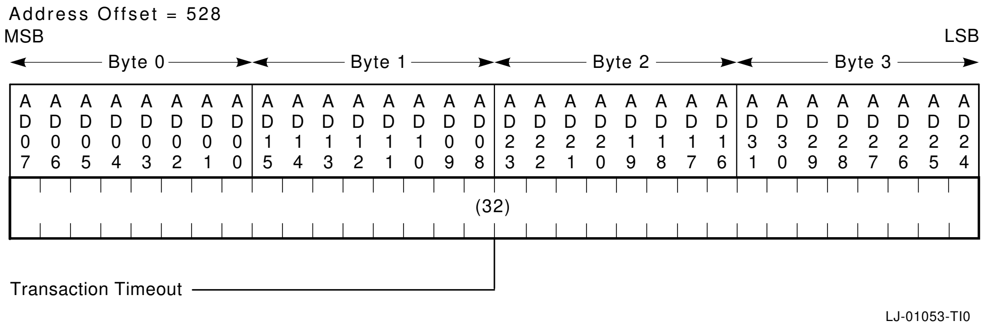
Field	Description	Access Type	Power Up	System Reset	Bus Initialize
PACKET_ ENABLE	If one, this bit specifies that the node has mode enabled.	R/W	0	0	0
64BIT_ ADDRESS	If one, this bit indicates that the node can use 64-bit addresses for locations accessible in 64-bit address space.	R/W	0	0	0
Number of TG[ ]* lines active	This field enables a specified number of TG[ ]* lines.	R/W	0	0	0
TAG_ENABLE	If this bit is one, it indicates that the number of tag bits specified are active and that the node must check tag parity.	R/W	0	0	0
PARITY_ REPORT_ ENABLE	The system configuration process sets this bit to enable the use of the status lines for signaling the detection of parity errors.	R/W	0	0	0
HIGH_SPEED_ ENABLE	The system configuration process sets this bit to indicate that the module can initiate and participate in high speed operations.	R/W	0	0	0
MASTER_ ENABLE	The system configuration process sets this bit to enable the master capability of the node. If zero, this node cannot initiate any transactions on the bus and should not split read transactions. If a transaction is split with this bit clear, a split TRANSACTION_TIMEOUT error is likely to occur because the node cannot become master to issue a Read Response transaction.	R/W	0	0	0

Continued on next page

TRANSACTION_TIMEOUT CSR

The TRANSACTION_TIMEOUT CSR format is shown in Figure 2–29.

Figure 2–29 TRANSACTION_TIMEOUT CSR Format



Continued on next page

Table 2–23 lists the TRANSACTION_TIMEOUT CSR bit definitions.

Table 2–23 TRANSACTION_TIMEOUT CSR Bit Definitions

Field	Description	Access Type	Power Up	System Reset	Bus Initialize
TRANSACTION_ TIMEOUT	Following a power up or system reset, this CSR will be set to 0x00080000 (122 μs). The field will be in steps of 2^{-32} seconds ($\approx$233 ps). All nodes in a module will have identical values in their TRANSACTION_ TIMEOUT CSRs. A value of zero will disable the transaction timeout.	R/W	0x00080000	0x00080000	0x00080000

ROM CSRs

Overview

The ROM CSRs are read-only registers that are specific to the IEEE 896 Futurebus+ standard. These CSRS are used to convey the capability information of the module and the individual nodes to the host CPU.

The initial register space of a node includes 1 kbyte of ROM at an address offset range of 1024 to 2044.

The ROM is used under normal operation as a storage location for read-only data or code. The information stored in the ROM is expected to be used by the system initialization and configuration process after a system reset, BUS_INIT, or live insertion. For example, the ROM will contain data concerning the capabilities of the node, the software and hardware version of the node, and the amount of node memory that may be mapped into Futurebus+ available address space. The overall format of the ROM is specified in the IEEE 1212 CSR Architecture (CSR Specification) standard.

The first entry in the ROM is specified to contain the ROM header as required by Profile B. Futurebus+ specifies or reserves an additional 15 locations of the ROM (each location is 4-bytes wide). The remaining 240 locations (960 bytes) are specified as available for root directory entries, unit subdirectories, root and unit leaves, and vendor-dependent information.

Continued on next page

Key Definitions

The following terms are used in the ROM.

Directory	A contiguous collection of one or more entries that are contained within the ROM of the node.
Leaf	A contiguous information field that is pointed to by a ROM-directory entry. A leaf contains a header (length and CRC specification) and other information fields.
Leaf entry	A ROM entry that specifies the address of a leaf.
Root directory	The directory at the top level of the hierarchical ROM directory structure.

Continued on next page

Futurebus+ ROM Directory Hierarchy

The Futurebus+ ROM hierarchy (shown in Figure 2–30), follows the IEEE 1212 CSR Architecture standard.

The ROM directory structure, according to IEEE 1212, is a hierarchy of information blocks; the blocks higher in the hierarchy point to the blocks beneath it.

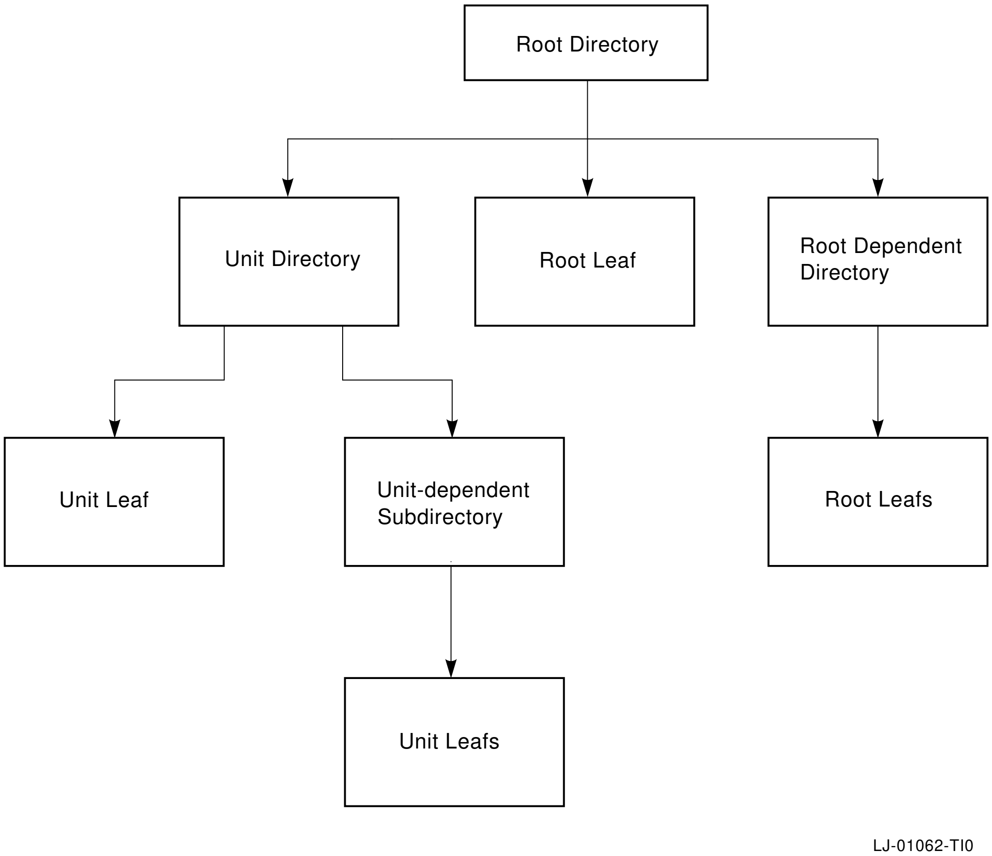
- The locations of the initial blocks (which include the Bus Information Block and the Root directory) are fixed.
- The offsets for information within the Bus Information Block are fixed.
- The offset for the start of the Root directory is fixed.
- The location of the various entries within the Root directory is not fixed.

  The key values in the Root directory entries describe the type of information to which the associated 24-bit fields refer. The only pointer type for which a location is (relatively) fixed is the TEXTUAL_DESCRIPTOR entry (0x81 or 0xC1). This type of entry always occurs immediately after the directory entry it describes. (A TEXTUAL_DESCRIPTOR pointer can follow any entry other than a TEXTUAL_DESCRIPTOR pointer.)

Each location can be an immediate value or an offset pointer to a leaf node or a directory structure. The most significant byte of each entry contains a key value that indicates how the entry should be interpreted.

Continued on next page

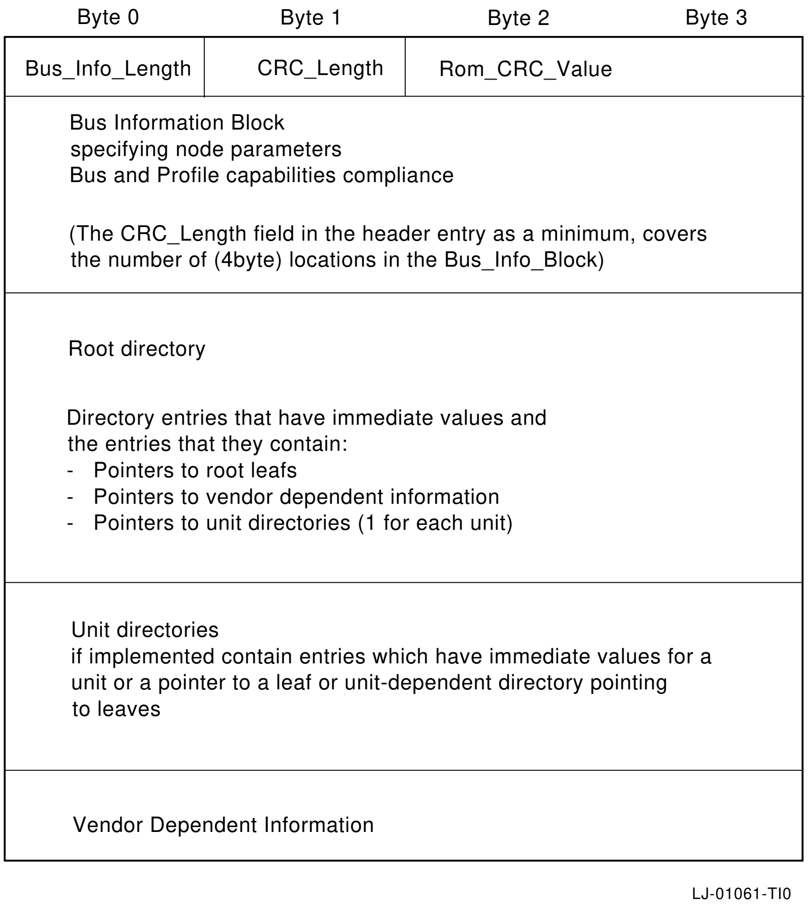
Figure 2–30 Futurebus+ ROM Directory Hierarchy



Continued on next page

Figure 2–31 shows the ROM Directory.

Figure 2–31 Futurebus+ ROM Directory



Continued on next page

Futurebus+ ROM Directory Hierarchy (continued)

The BUS_INFO_LENGTH field specifies the number of (4-byte) entries contained in the BUS_INFO_BLOCK (not including the header).

The CRC_LENGTH field specifies how many of the following locations are protected by the ROM_CRC_Value:

- The minimum number of protected locations is the number within the Bus Information Block
- The maximum is 255, which covers the entire Initial Futurebus+ ROM address space.

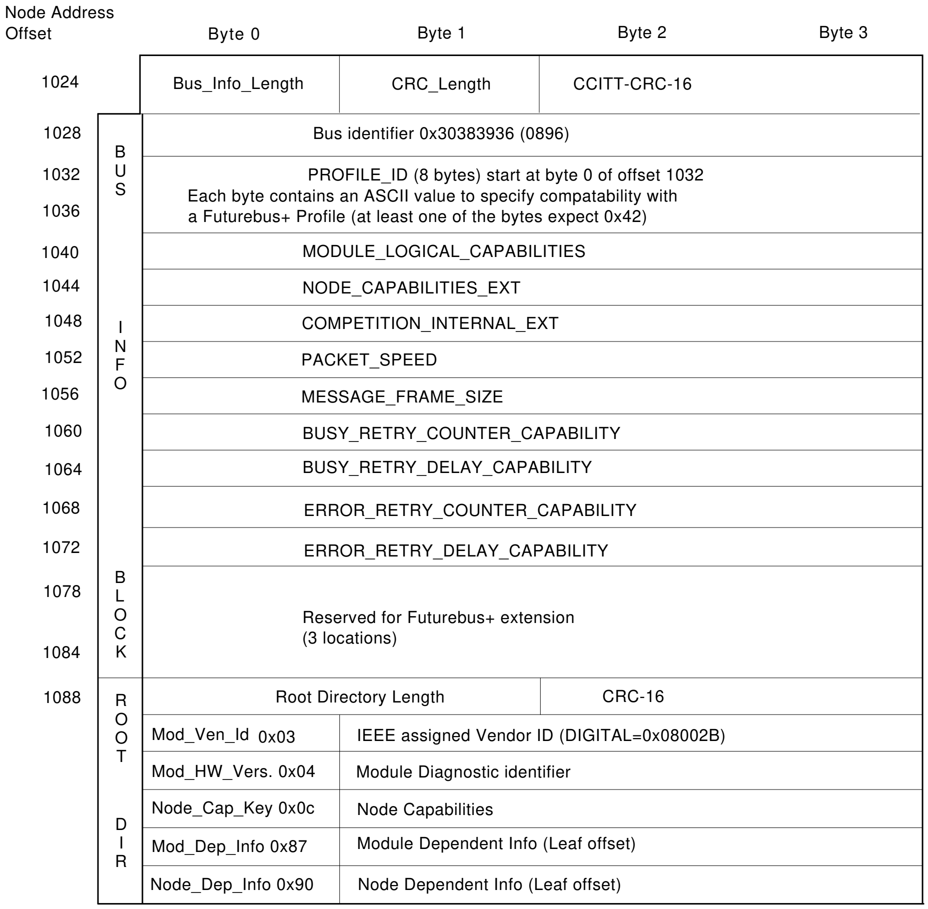
The Root Directory of the ROM contains entries that describe the module, node, and unit or units. The area also provides pointers to additional information contained in the ROM, such as other subdirectories (Unit Subdirectory block) or leaves (Root and Unit Leaf blocks) that contain the actual data.

The last portion of the ROM structure is the vendor dependent information block. This block contains parameters that may vary among vendors supplying the same node or unit architectures.

Figure 2–32 shows the ROM structure in detail.

Continued on next page

Figure 2–32 Detailed ROM Structure



Notes:

- For Futurebus+ the Bus_Info_Length will always be 0xf.
- The CSR entries in the Bus Info Block are found at the fixed offsets shown and described in Section 3 of the IEEE 896.2.
- The Root Directory structure for Futurebus+ always starts at offset 1088; the entries are not at fixed offsets.

LJ-01065-TI0

Continued on next page

ROM Header

The first 4-byte entry in the ROM starting at node offset 1024 (shown in Figure 2–32) contains three fields. The first field (INFO_LENGTH) is used to indicate the number of 4-byte locations contained in the Bus Information Block. The INFO_LENGTH field will always contain a value of 15 on Futurebus+ systems. The CRC_LENGTH field indicates the number of 4-byte locations protected by the value in the CRC field. For example, if the length field contained 0xFF (255), the entire ROM (except for the first location) would be protected by the value in the CRC field.

Continued on next page

Bus Information Block

The ROM Bus Information Block contains information that identifies the module as a Futurebus+ product, lists application environment profile compatibility data, and contains bus-dependent ROM entries. It also gives the system key information as to what types of operations the module is capable of performing.

The following is a list of Profile B required CSRs that are contained in the Bus Information block:

- Bus Identifier
- Profile ID
- MODULE_LOGICAL_CAPABILITY
- NODE_CAPABILITIES_EXT
- COMPETITION_INTERNAL_DELAY
- PACKET_SPEED
- MESSAGE_FRAME_SIZE
- BUSY_RETRY_COUNTER_CAPABILITY
- BUSY_RETRY_DELAY_CAPABILITY
- ERROR_RETRY_COUNTER_CAPABILITY
- ERROR_RETRY_DELAY_CAPABILITY

Bus Identifier

This 4-byte field, starting at node offset 1028, identifies the module as Futurebus+ by its IEEE standard number of 0896. This number is enumerated in ASCII: Bus Identifier = 0x30383936 (ASCII "0896").

Continued on next page

PROFILE_ID

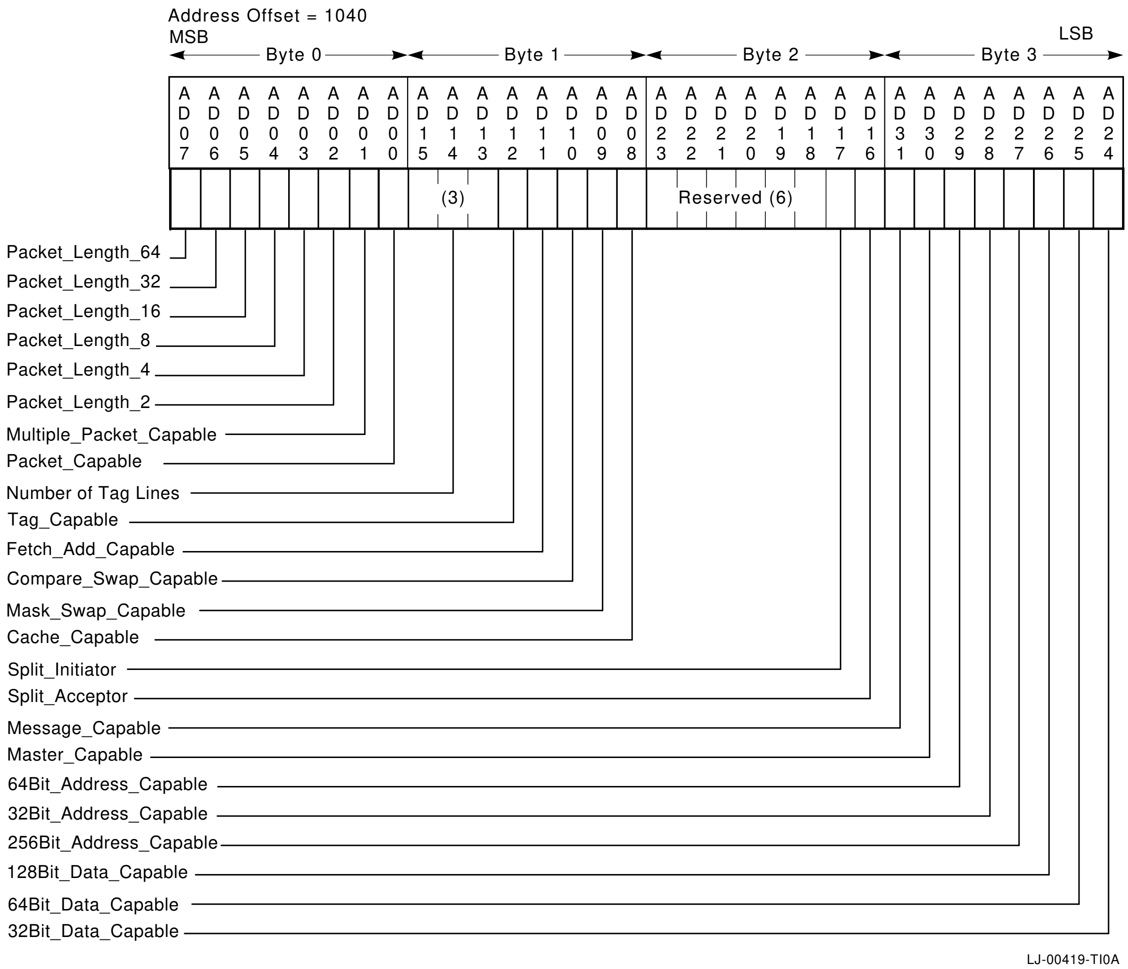
This 8-byte field starting at node offset 1032, identifies the profile or profiles to which the module vendor claims compliance. At least 1 byte of this field is expected to be 0x42 (ASCII "B"), to indicate that it is compliant with Profile B.

Continued on next page

MODULE_LOGICAL_CAPABILITY CSRs

The MODULE_LOGICAL_CAPABILITY CSR format is shown in Figure 2–33.

Figure 2–33 MODULE_LOGICAL_CAPABILITY CSR Format



Continued on next page

Table 2–24 lists the MODULE_LOGICAL_CAPABILITY bit definitions.

Table 2–24 MODULE_LOGICAL_CAPABILITY CSR Bit Definitions

Field	Description	Access Type
	Most Significant Half	
PACKET_ LENGTH_n	These bits indicate which lengths the module supports. Legal lengths are 2, 4, 8, 16, 32, and 64.	RO
MULTIPLE_ PACKET_ CAPABLE	This bit is set if the module is capable of participating in transactions using the protocol in multiple mode.	RO
PACKET_ CAPABLE	This bit is set if the module is capable of participating in transactions using the protocol.	RO
Number of Tag lines	This 3-bit field contains the number of TG[ ]* lines a module has implemented. A value of zero corresponds to one TG[ ]* line, and a value of seven corresponds to eight TG[ ]* lines.	RO
TAG_CAPABLE	This bit is set if the module has one or more TG[ ]* lines implemented.	RO
FETCH_ADD_ CAPABLE	This bit indicates if the module is capable of responding to FETCH_ADD locked commands.	RO
COMPARE_ SWAP_ CAPABLE	This bit indicates if the module is capable of responding to COMPARE_SWAP locked commands.	RO
MASK_SWAP_ CAPABLE	This bit indicates if the module is capable of responding to MASK_SWAP locked commands.	RO
CACHE_ CAPABLE	This bit indicates if the module is capable of participating in transactions to cache-coherent shared memory.	RO

Continued on next page

Table 2–24 (Continued) MODULE_LOGICAL_CAPABILITY CSR Bit Definitions

	Least Significant Half	
SPLIT_ INITIATOR	This bit indicates that a module is capable of initiating split transactions. A split initiator is capable of asserting SR* and later performing a response bus transaction.	RO
SPLIT_ ACCEPTOR	This bit indicates that a module is capable of accepting split transactions. When a split acceptor is master and performing a transaction, it is capable of detecting SR* asserted, ending its transaction, and waiting for a response transaction from the module that asserted SR*.	RO
MESSAGE_ CAPABLE	This bit indicates that the module is capable of participating in message-passing transactions, as specified in IEEE 896.1, Chapter 9.	RO
MASTER_ CAPABLE	This bit indicates that a module is capable of bus mastership.	RO
64BIT_ ADDRESS_ CAPABLE	This bit indicates that the module can generate a 64-bit address width, in addition to the default 32-bit address width.	RO
32BIT_ ADDRESS_ CAPABLE	This bit indicates that the module can generate a 32-bit address and be set on all nodes.	RO
256BIT_DATA_ CAPABLE	This bit indicates that the module supports a 256 bit data width.	RO
128BIT_DATA_ CAPABLE	This bit indicates that the module supports a 128 bit data width.	RO
64BIT_DATA_ CAPABLE	This bit indicates that the module supports a 64 bit data width.	RO
32BIT_DATA_ CAPABLE	This bit indicates that the module supports a 32 bit data width.	RO

NOTE: All modules support the default 32-bit data width.

Continued on next page

NODE_CAPABILITIES_EXT CSR

The NODE_CAPABILITIES_EXT CSR format is shown in Figure 2–34.

Figure 2–34 NODE_CAPABILITIES_EXT CSR Format

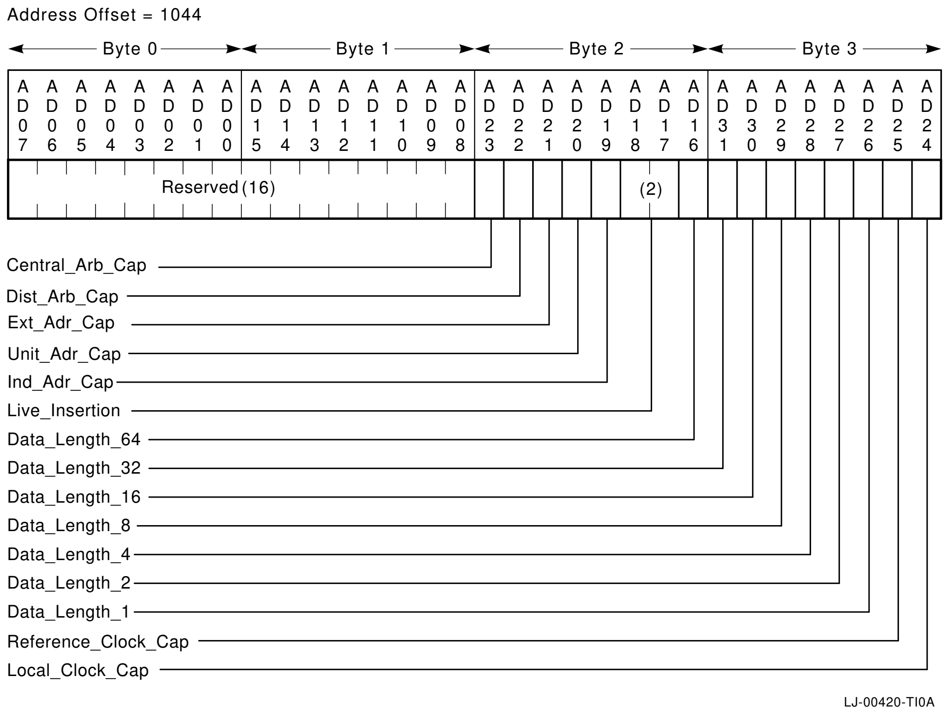


Table 2–25 lists the NODE_CAPABILITIES_EXT CSR bit definitions.

Continued on next page

Table 2–25 NODE_CAPABILITIES_EXT CSR Bit Definitions

Field	Description	Access Type
CENTRAL_ ARB_CAP	If one, this bit indicates that this node supports central arbitration. If zero, central arbitration is not supported. All Profile B modules must support central arbitration.	RO
DIST_ARB_CAP	If one, this bit indicates that this node supports distributed arbitration.	RO
EXT_ADR_CAP	If one, this bit indicates that this node supports external address space capability implemented with the MEMORY_BASE and MEMORY_BOUND CSRs.	RO
UNIT_ADR_CAP	If one, this bit indicates that this node supports the units address space capability implemented with the UNITS_BASE and UNITS_BOUND CSRs.	RO
IND_ADR_CAP	If one, this bit shall indicate that this unit supports the address space capability implemented with the INDIRECT_ADDRESS and INDIRECT_DATA CSRs.	RO

Continued on next page

Table 2–25 (Continued) NODE_CAPABILITIES_EXT CSR Bit Definitions

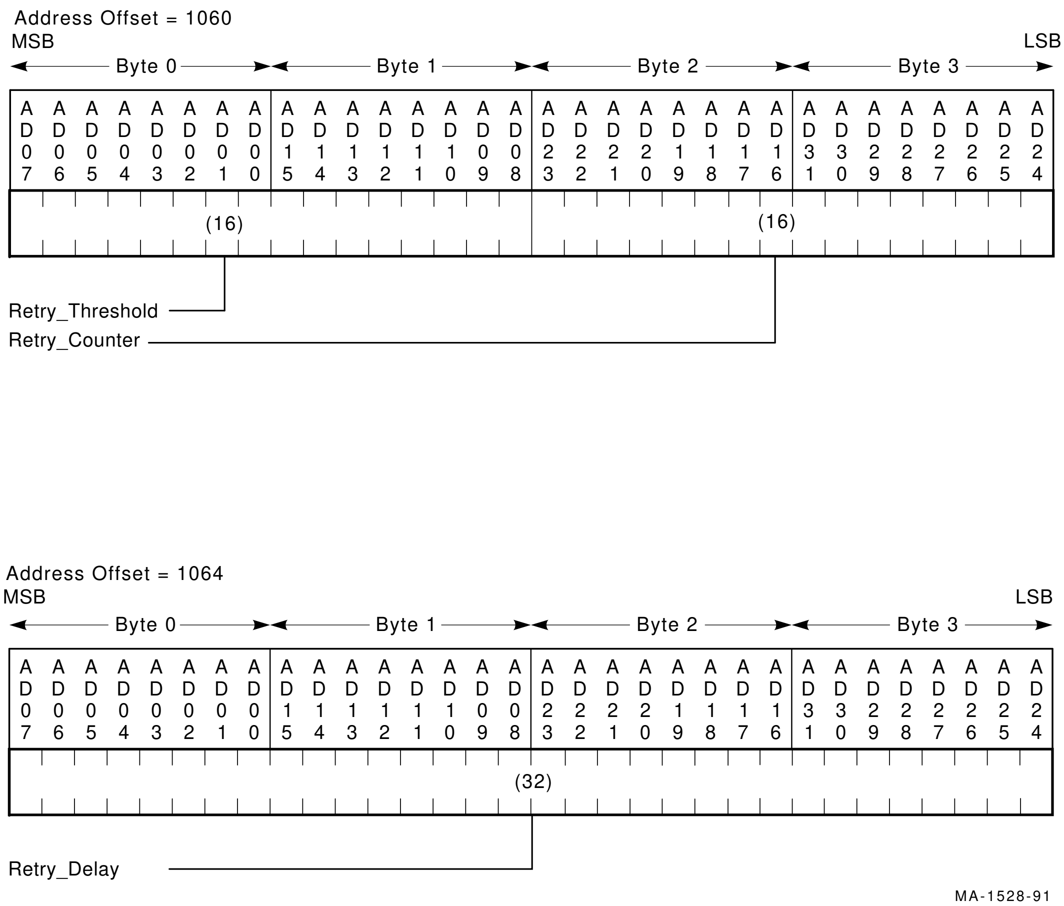
Field	Description	Access Type
LIVE_ INSERTION	This 2-bit field indicates the level of live insertion supported by this module. If a module contains two nodes, this field will be the same in both modules. The live insertion levels are specified in IEEE 896.2, Chapter 4. **Field Value** **Decimal / Binary / Description** 0 / 0b00 / Does not support live insertion (level 0). 1 / 0b01 / Supports level 1 live insertion. 2 / 0b10 / Supports level 2 live insertion. 3 / 0b11 / This module supports live insertion without constraints (level 3).	RO
DATA_ LENGTH_n	These 7 bits indicate which compelled data transfer lengths the module supports. Values supported are transfer lengths of 1, 2, 4, 8, 16, 32, and 64. These values are multiples of a data beat.	RO
REFERENCE_ CLOCK_CAP	If one, this bit indicates that if the node contains a timer, the timer can function as the reference timer for a Futurebus+ system.	RO
LOCAL_ CLOCK_CAP	If one, this bit indicates that if the node contains a timer, the timer can function as a subordinate timer for a Futurebus+ system.	RO

Continued on next page

BUSY_RETRY_CAPABILITY CSRs

The BUSY_RETRY_COUNTER_CAPABILITY and BUSY_RETRY_DELAY_CAPABILITY CSR formats are shown in Figure 2–35.

Figure 2–35 BUSY_RETRY_CAPABILITY CSR Format



Continued on next page

Table 2–26 lists the BUSY_RETRY_COUNTER_CAPABILITY and Table 2–27 lists the BUSY_RETRY_DELAY_CAPABILITY CSR bit definitions.

Table 2–26 BUSY_RETRY_COUNTER_CAPABILITY CSR Bit Definitions

Field	Description	Access Type
RETRY_ THRESHOLD	This bit field is read by the system initialization and configuration process to determine the values to be loaded into the BUSY_ RETRY_COUNTER CSRs of the various nodes. If this field contains 0x0000, this node does not support busy retries.	RO
RETRY_ COUNTER	This field must contain 0x0000 for the ROM capability CSR.	RO

Table 2–27 BUSY_RETRY_DELAY_CAPABILITY CSR Bit Definitions

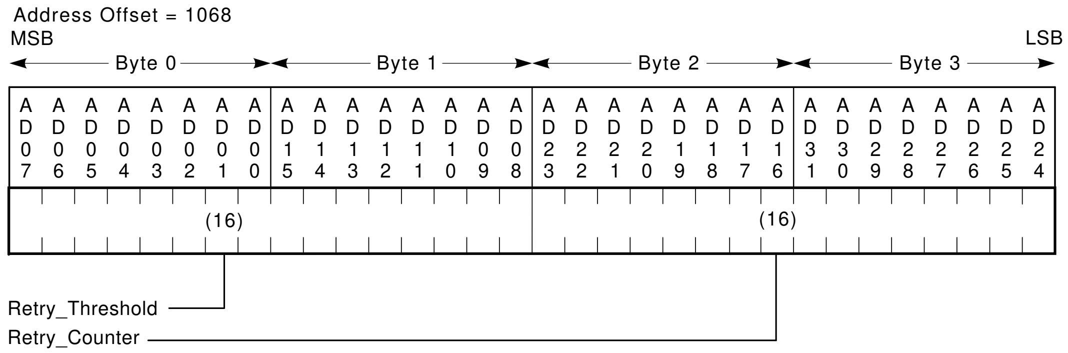
Field	Description	Access Type
RETRY_ DELAY	This time value is in units of 2^{-32} seconds (≈233 ps). A value of 0x00000000 indicates the operation should be retried as soon as possible.	RO

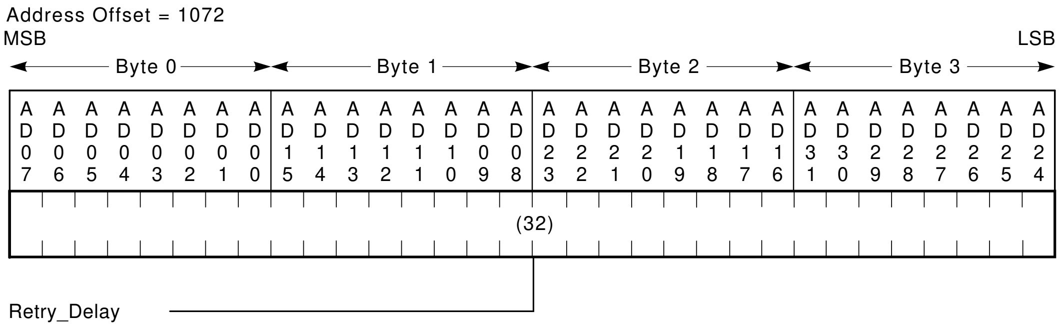
Continued on next page

ERROR_RETRY_CAPABILITY CSRs

Figure 2–36 shows the ERROR_RETRY_COUNTER_CAPABILITY and ERROR_RETRY_DELAY_CAPABILITY CSR formats.

Figure 2–36 ERROR_RETRY_COUNTER_CAPABILITY and ERROR_RETRY_DELAY_CAPABILITY CSR Formats





Continued on next page

Table 2–28 lists the ERROR_RETRY_COUNTER_CAPABILITY and Table 2–29 lists the ERROR_RETRY_DELAY_CAPABILITY CSR bit definitions.

Table 2–28 ERROR_RETRY_COUNTER_CAPABILITY CSR Bit Definitions

Field	Description	Access Type
RETRY_ THRESHOLD	This bit field is read by the system initialization and configuration process to determine the values to be loaded into the ERROR_ RETRY_COUNTER CSRs of the various nodes. If this field contains 0x0000, this node does not support error retries.	RO
RETRY_ COUNTER	This field must contain 0x0000 for the ROM capability CSR.	RO

Table 2–29 ERROR_RETRY_DELAY_CAPABILITY CSR Bit Definitions

Field	Description	Access Type
RETRY_ DELAY	This time value is in units of 2^{-32} seconds ($\approx$233 ps). A value of 0x00000000 indicates the operation should be retried as soon as possible.	RO

Continued on next page

Root Directory

As mentioned in previous sections, the first part of ROM is the header, the next part is the bus dependent area and has information at fixed, architecturally specified offsets. The third section of the ROM is called the Root Directory. Figure 2–32 shows the following entries in this section:

- Root Directory Length and CRC
- MODULE_VENDOR_ID
- MODULE_HW_VERSION
- NODE_CAPABILITIES
- MODULE_DEPENDENT_INFO
- NODE_DEPENDENT_INFO
- Other Root Directory Entries

ROM Directory Descriptors

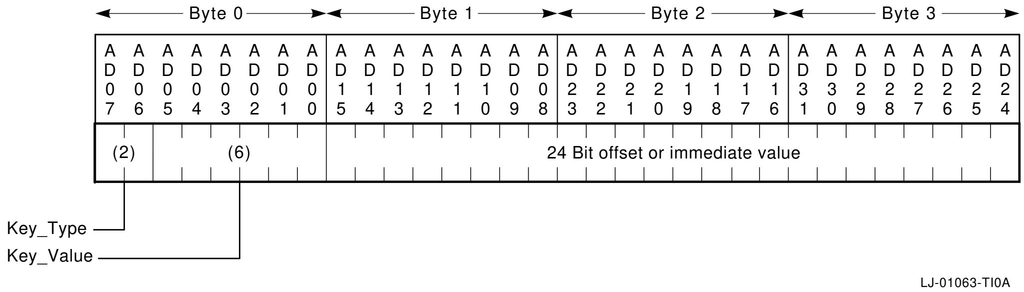
The ROM directory format entries are 32-bit directory descriptors. The value of the 8-bit key field specifies:

- The nature of the 24-bit entry value (immediate or offset)
- The address space for which the offset is specified is either of the following:
  - — Initial node space for CSR offsets
  - — Indirect node space for leaf or directory offsets
- The type of directory entry it contains or points to (indicated by 6-bit key value)

Continued on next page

Figure 2–37 shows the ROM directory descriptor field.

Figure 2–37 Futurebus+ ROM Descriptor Entry Format



Continued on next page

Table 2–30 lists the key type and entry values for a 2-bit field.

Table 2–30 Key Types

Type Decimal	Type Binary	Meaning of Entry Value
0	0b00	Immediate value
1	0b01	CSR-offset
2	0b10	Indirect-space offset to a leaf entry
3	0b11	Indirect-space offset to a lower level directory

- The key and immediate values are defined in the IEEE 1212 CSR Architecture standard, ROM chapter.
- The 24-bit immediate values are defined for each individual key value that uses immediate type keys.
- The CSR offset is a quadlet offset from the initial address space of a node, which is used to calculate a CSR address.
- The leaf and directory offsets are indirect address space offsets used to calculate an address in the indirect address space.
- The 24-bit indirect offset has the following relationship for indirect node address space offsets that appear in ROM CSR space:

```
indirect_address_of_item_pointed_to = (CSR address of the
pointing entry - 1024) + 4 * indirect offset entry
```

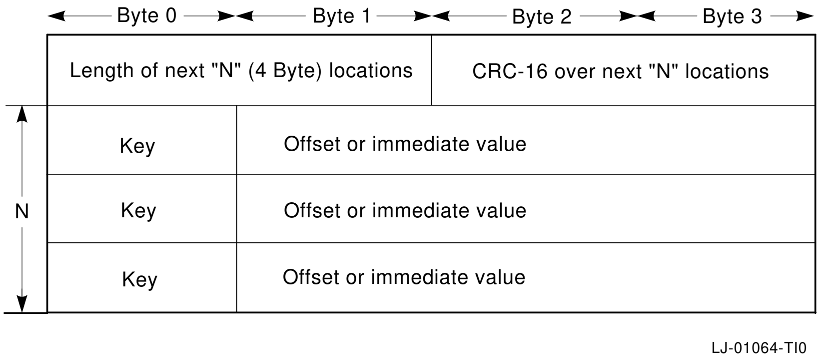
Continued on next page

Subdirectory and Leaf Format

All subdirectories and leaf entries have the same header format, as shown in Figure 2–38. The only difference between the directory and the leaf is the format of the entries.

In directories, there are keyed directory style entries that, depending on the key type, can be immediate or offset values. The leaf entries are vendor-dependent. The contents of the minimum implementation for vendor-dependent entries of Digital Futurebus+ options or modules are defined in this handbook. Figure 2–38 shows the Futurebus+ ROM Root directory, subdirectory, and leaf format.

Figure 2–38 Futurebus+ ROM Root Directory, Subdirectory, and Leaf Format



In Figure 2–38, the length field is 3. The CRC-16 value covers the next 3 locations.

The leaf header uses the same format and method for length and CRC-16 determination as the Root directory and lower-level directory header; however, the entries following the length /CRC-16 entry contain vendor-defined data.

Continued on next page

CRC-16 Algorithm

The CRC-16 algorithm is based on CCITT CRC-16 code that is performed on the most significant bits first. This algorithm is specified in the IEEE 1212 CSR Architecture standard, ROM chapter, and in Appendix A of this handbook.

MODULE_VENDOR_ID

This ROM entry identifies the vendor manufacturing the module. It is identified by a key value of 0x03 in the most significant byte. It is the same (24-bit) number as assigned by the ISO 8802-3 (Ethernet) registration authority.

MODULE_SPEC_ID

This field allows second sources of a specified architecture. If the module was designed to a specification or standard defined by a vendor other than that identified in the MODULE_VENDOR_ID, then the ISO 802-3 number associated with the vendor that specified the design appears here.

This entry is identified by a key value of 0x05, in the most significant byte. If not implemented, the assumed value is equal to the MODULE_VENDOR_ID value.

MODULE_HW_VERSION

This field is used to identify the diagnostic software for extensive module tests. It is the vendor's responsibility to provide a unique identifier.

Continued on next page

NODE_CAPABILITIES Entry Format

The NODE_CAPABILITIES Entry CSR format is shown in Figure 2–39.

Figure 2–39 NODE_CAPABILITIES Entry

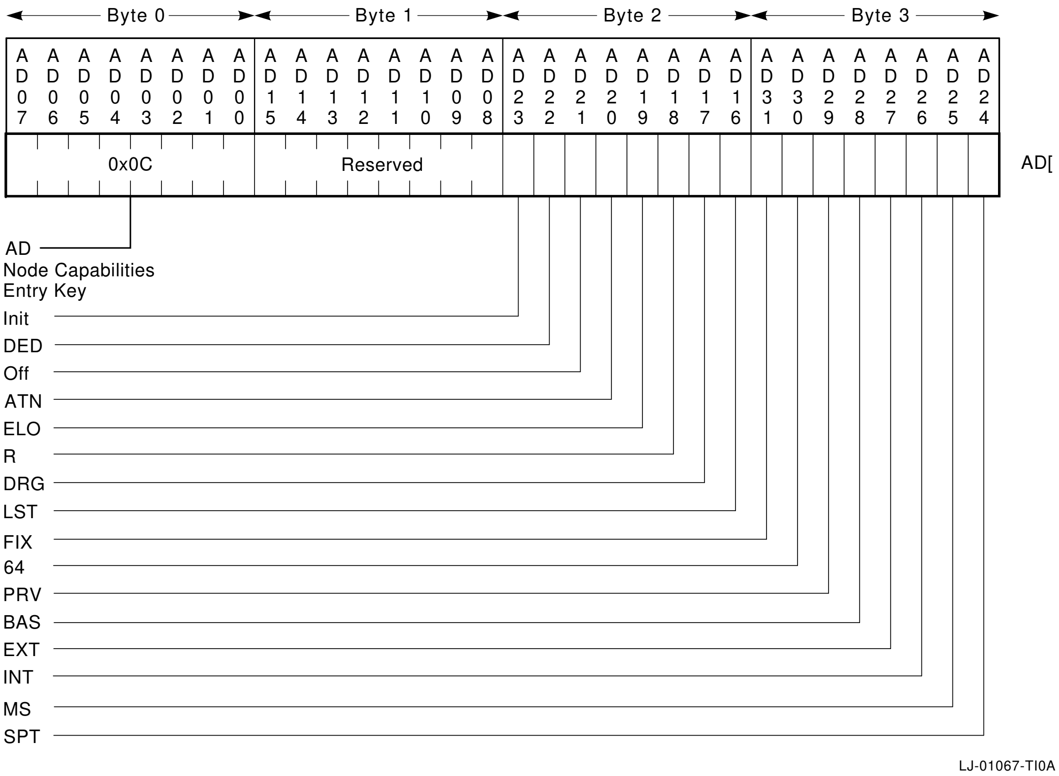


Table 2–31 lists the NODE_CAPABILITIES entries.

Continued on next page

Table 2–31 Bit Positions

Group	Bit	Description	Access Type
State	INIT	One indicates the node supports the initializing state.	RO
	DED	One indicates the node supports the dead state.	RO
	OFF	One indicates the STATE_BITS.OFF bit is implemented.	RO
	ATN	One indicates the STATE_BITS.ATN bit is implemented.	RO
	ELO	One indicates the STATE_BITS.ELO bit and the error log buffer are implemented.	RO
	R	This bit is reserved by IEEE 1212.	RO
	DRQ	One indicates the STATE_BITS.DREQ bit is implemented.	RO
	LST	One indicates the STATE_BITS.LOST bit is implemented.	RO
Addressing	FIX	One indicates the node uses the fixed addressing scheme; otherwise, the node uses the extended addressing scheme.	RO
	64	One indicates the node uses 64-bit addressing; otherwise, the node uses 32-bit addressing.	RO

Continued on next page

Table 2–31 (Continued) Bit Positions

Group	Bit	Description	Access Type
	PRV	One indicates this node implements the private space.	RO
Testing	BAS	One indicates that the node is capable of basic testing, and the TEST_START and TEST_STATUS CSRs are implemented.	RO
	EXT	One indicates that the node is capable of extended testing, and the ARGUMENT CSRs are implemented.	RO
Misc.	INT	One indicates that INTERRUPT_TARGET and INTERRUPT_MASK CSRs are implemented.	RO
	MS	One indicates that the MESSAGE_PASSING CSRs are implemented.	RO
	SPT	One indicates that the SPLIT_TIMEOUT CSR is implemented.	RO
Reserved	reserved	Reserved by IEEE 1212.	RO

Continued on next page

MODULE_DEPENDENT_INFO

The MODULE_DEPENDENT_INFO entry leaf contains vendor-dependent data about the module.

Vendor Dependent ASCII Module Description

This is an ASCII description of the function of the module. This field may be printed on the console, entered in an error log, or ignored.

This field is required by Profile B; however, it should not be used as the only basis for the selection of the device driver to be loaded.

Profile B Minimum Required Root Directory Entries

The Root Directory of the ROM contains entries that describe the module, node, and unit(s). The area also provides pointers to additional information contained in the ROM, such as other subdirectories (Unit Subdirectory block) or leafs (Root and Unit Leaf blocks) that contain the actual data.

Table 2–32 shows the ROM Directory entries with their respective key values for use in the Profile B Futurebus+ ROM node space implementation.

Table 2–32 ROM Directory Required Entries

8-bit Key Value	Key Type	Root Directory Entry Descriptors
0x03	Immediate	MODULE_VENDOR_ID This key indicates that the immediate value provides the company ID of the vendor manufacturing the module.
0x06	Immediate	MODULE_SW_VERSION

Continued on next page

Table 2–32 (Continued) ROM Directory Required Entries

8-bit Key Value	Key Type	Root Directory Entry Descriptors
		This key specifies that the immediate value is used to identify the I/O driver for this module. Do not specify this entry if NODE_SW_VERSION or UNIT_SW_VERSION is specified.
0x0B	Immediate	NODE_SW_VERSION
		This key specifies that the immediate value is used to identify the I/O driver for this node. Do not specify this entry if MODULE_SW_VERSION or UNIT_SW_VERSION is specified.
0x0C	Immediate	NODE_CAPABILITIES
		This key specifies that the bits in the immediate entry indicate which options in the IEEE 1212 CSR Specification are implemented in this node.
0x87	Leaf Offset	MODULE_DEPENDENT_INFO
		This key specifies that the offset points to a leaf with module vendor-dependent data.
0x90	Leaf Offset	NODE_DEPENDENT_INFO
		This key specifies that the offset points to a leaf with node vendor-dependent data.

The IEEE 1212 CSR standard specifies the following three types of software version keys. These entries provide identifiers for associating the correct I/O drivers with a particular module, node, or unit. The standard details a software identifier hierarchy concatenating MODULE_VENDOR_ID or NODE_VENDOR_ID entry values with the respective _SW_VERSION entry values to uniquely identify associated drivers.

Continued on next page

Each key entry is mutually exclusive; that is, only one software version key entry can be present in the ROM space of a node.

- MODULE_SW_VERSION key

  This key is used if there is only one I/O driver for module. This key is specified in the node Root Directory.

- NODE_SW_VERSION key

  This key is used for module with 2 nodes, each node with its own I/O driver. This key is specified in the Root Directory of each node.

- UNIT_SW_VERSION key

  This key is used when Unit Directories are implemented. The key is also used to identify the I/O driver for the unit, and is specified in the Unit Directory for each unit.

Table 2–33 lists the ROM Directory entries for optional implementations.

Table 2–33 Root Directory Entries for Optional Implementations

8-bit Key Value	Key Type	Root Directory Entry Descriptors
0xD1	Dir offset	UNIT_DIRECTORY (optional) This key specifies an offset pointer to a Unit Directory. If implemented, there will be an entry pointing to a Unit Directory for each unit. If UNIT_DIRECTORY keys are implemented, neither MODULE_SW_VERSION or NODE_SW_VERSION entries can be present according to the IEEE 1212 CSR Specification.
0x04	Immediate	MODULE_HW_VERSION (required)

Continued on next page

Table 2–33 (Continued) Root Directory Entries for Optional Implementations

8-bit Key Value	Key Type	Root Directory Entry Descriptors
		This key is used to identify the diagnostic software for extensive module tests or identification for IVP at installation verification

Other Root Directory and Unit Directory key values are defined in the IEEE 1212 CSR standard and summarized in Appendix B of this handbook.

Unit Directory

The Unit_Directory (0xD1) entries in the Root Directory point to the Unit Directory or directories, one for each unit on a node if implemented.

The IEEE 1212 CSR Specification defines the entries that are required if an (optional) unit subdirectory is implemented. These entries are listed in Table 2–34.

Table 2–34 Recommended Minimum and Required Entry Implementation

8-bit Key Value	Key Type	Directory Entry Descriptors
0x13	immediate	UNIT_SW_VERSION This key identifies the I/O driver for this unit.
0x94	leaf offset	UNIT_DEPENDENT_INFO

Continued on next page

Table 2–34 (Continued) Recommended Minimum and Required Entry Implementation

8-bit Key Value	Key Type	Directory Entry Descriptors
		This key points to a leaf that contains unit-specific information such as partnumber, serial number, unit firmware revision level, and so on.
0x95	leaf offset	UNIT_LOCATION This key is required if two or more Unit Directories are present. This entry points to a leaf that provides this unit's address mapping information to a range of addresses within the node Initial Units address space.

Continued on next page

NODE_DEPENDENT_INFO Leaf Format

The NODE_DEPENDENT_INFO leaf provides information about a particular node on the module. The information provided is vendor-dependent.

UNIT_SW_VERSION Entry

Refer to the MODULE_SW_VERSION entry described in the previous section of this chapter.

UNIT_LOCATION Leaf

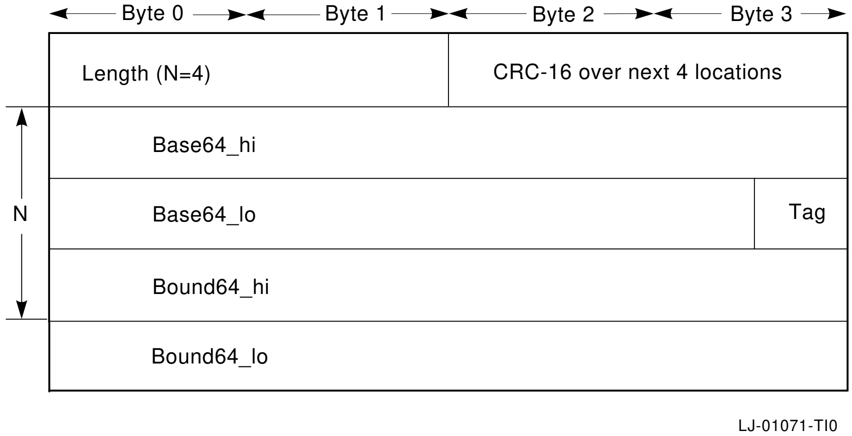
The IEEE 1212 CSR standard dictates that the UNIT_LOCATION leaf be a required entry if two or more Unit Directories are present. This entry is pointed to by the UNIT_LOCATE key entry (0x95) in the Unit Directory.

The UNIT_LOCATION leaf provides two 64-bit address values: a boundary (= last unit address +1) that specifies the Unit's map of contiguous addresses within the node address space; and a 2-bit tag field to indicate where the address space mapped by base and boundary values is located.

Figure 2–40 shows the UNIT_LOCATION leaf entry format.

Continued on next page

Figure 2–40 UNIT_LOCATION Leaf



The value of the 2-bit tag field (LSB+1, LSB) defines where the address space is located.

Tag	Unit Register Location
00	Initial register space
01	Initial memory or extended units space
10	Indirect space
11	Reserved

Continued on next page

UNIT_DEPENDENT_INFO Leaf

The UNIT_DEPENDENT_INFO leaf provides information about a particular unit on the module. The information provided is vendor-dependent.

Initial Unit Space Registers

Initial Unit Space Registers

The Futurebus+ Profile B does not mandate the use of any Initial Unit Space CSRs. However, in most cases a node will need some Initial Unit Space CSRs for various reasons. For example, any internal module CSRs may be mapped into Initial Unit Space. Addresses 2048-4092 are reserved for this purpose.

Chapter 3
Profile B Logical Layer

Logical Layer Overview

This chapter provides an overview of the Futurebus+ Profile B logical layer. The logical layer is the timing and behavior of the set of lines that are specified in the physical layer of the Futurebus+.

This chapter describes two major areas:

- The individual lines that comprise the Futurebus+
- The various operations that employ these lines

Initialization

Any of the following methods can be used to initialize a Profile B module:

- System Reset
- Power Up
- Node Directed Reset
- Bus Initialization

Continued on next page

Logical Layer Overview, Continued

Arbitration

Digital systems use a central arbitration protocol to give access to the bus. The protocol allows high-speed operation and low request-to-grant latency.

Supported Transactions

Profile B modules transfer data using the parallel protocol. These modules support the following transactions.

- Read Unlocked
- Read Partial
- Read Response transaction
- Write Unlocked
- Write Partial
- Address Only Unlocked

Continued on next page

Unsupported Transactions

Profile B modules support a subset of the Futurebus+ protocol. These modules *do not* support the following transactions:

- Address Only Locked
- Copyback
- Invalidate
- Modified Response
- Read Invalid
- Read Locked
- Read Modified
- Read Partial Locked
- Read Shared
- Shared Response
- Write Invalid
- Write Locked
- Write No Acknowledge
- Write Partial Locked
- Write Response

Bus Lines and Signals: Terms

Bus Line

A **bus line** is the medium used to transmit signals. Because Futurebus+ requires drivers with wired-OR capability, a bus line may be driven by several modules simultaneously. Therefore, the signal carried by the bus line is the logical "OR" of signals applied to that line from each module.

Activate

To **activate** is to apply a set of signals to a group of bus lines. The term is also used to describe the state of a group of bus lines when they carry signals. An asterisk (*) after a signal name indicates that the logic one state of the signal will override the logic zero state applied by any other module on that line.

Assert

To **assert** is to apply a logic one signal to a bus line. The term is also used to describe the state of a bus line when the signal it carries represents a logic one. In the case of asserted signals, the voltage seen on the bus is approximately 1.0 V.

Release

To **release** is to apply a logic zero signal to a bus line. The term is also used to describe the state of a bus line when the signal it carries represents a logic zero. In the case of released signals, the voltage seen on the bus is approximately 2.1 V.

Signal Names

Groups of related bus lines are given names such as AD*, RQ*, and so forth. Individual lines within the group are numbered, for example, AD0*, AD1*, AD2*. To represent a group of lines or signals in more convenient form, the notation AD[63..0]* is used. The notation AD[ ]* is used frequently to refer to all of the lines within the group. Additionally, within the IEEE standards, variations on the signal name are used to indicate a specific version of the signal. Figure 3–1 shows the signal naming conventions; Table 3–1 provides details.

Continued on next page

Signal Names (continued)

Figure 3–1 Signal Conventions

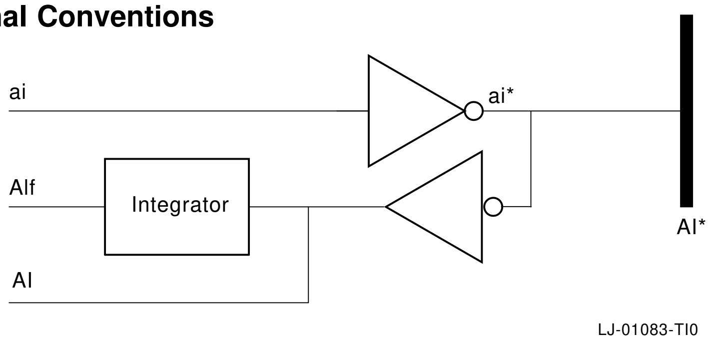


Table 3–1 Conventional Figure Labels

Labels	Meaning
ai	A specific module applies this version of the signal to the input of its driver.
ai*	A specific module applies this version of the signal to a bus line.
AI*	The signal that appears on a bus line as the result of the combined signals applied to it by all of the modules and the bus terminators.
AI	The signal AI after it has passed through an inverting receiver from the AI* bus line.
AIf	An "f" added to a signal name refers to the bus line signal after it has passed through a receiver and a wired-OR glitch filter (integrator).

Bus Lines and Signals: Overview

Signal/Line Description

The bus lines include the following major groups:

- Reset/initialization
- Geographical address
- Central arbitration
- Arbitration bus (distributed arbitration)
- Parallel protocol synchronization
- Parallel protocol information
- Serial bus

Table 3–2 lists the signals, line names, and data path widths for the bus signals required to support Futurebus+ Profile B. Each line is described in detail in a subsequent section of this chapter.

Table 3–2 Bus Lines

Signals	Line Name	Data Path Width		
Reset Initialization Line		**32-bit**	**64-bit**	**128-bit**
RE*	Reset	1	1	1
Geographical Address Lines		**32-bit**	**64-bit**	**128-bit**
GA[4..0]*	Geographical Address	5	5	5
Central Arbitration Lines		**32-bit**	**64-bit**	**128-bit**
RQ0*	Request Low	1	1	1
RQ1*	Request High	1	1	1
GR*	Grant	1	1	1
PE*	Preempt	1	1	1
Arbitration Bus (Dist. Arb.) Lines		**32-bit**	**64-bit**	**128-bit**

Continued on next page

Table 3–2 (Continued) Bus Lines

Signals	Line Name	Data Path Width		
AB[7..0]*[1]	Arbitratration Bus	8	8	8
ABP*[1]	Arbitration Bus Parity	1	1	1
AP*[1], AQ*[1], AR*[1]	Arbitratration Bus Synchronization	3	3	3
AC[1..0]*[1]	Arbitration Bus Condition	2	2	2
Parallel Protocol Synchronization Lines		**32-bit**	**64-bit**	**128-bit**
AS*, AK*, AI*	Address Handshake	3	3	3
DS*, DK*, DI*	Data Handshake	3	3	3
ET*	End of Tenure	1	1	1
Parallel Protocol Information Lines		**32-bit**	**64-bit**	**128-bit**
AD[63..0]*	Address/Data	32	64	64
D[265..64]*	Data	-	-	64
BP[31..0]*	Bus Parity	4	8	16
CM[7..0]*	Command	8	8	8
CP*	Command Parity	1	1	1
ST[7..0]*	Status	8	8	8
CA[2..0]*[2]	Capability	3	3	3
TG[7..0]*[3]	Tag	8	8	8
TP*[3]	Tag Parity	1	1	1
Serial Bus Lines		**32-bit**	**64-bit**	**128-bit**
SB0*[4]	Serial Bus 0	1	1	1

[1]Not needed on an I/O module if power fail messages are not decoded.

[2]CA0* (TS*) may be optionally deleted if packet mode is not supported.

[3]Not needed if tags are unused.

[4]Not needed if the serial bus is unused.

Continued on next page

Table 3–2 (Continued) Bus Lines

Signals	Line Name	Data Path Width		
SB1*[4]	Serial Bus 1	1	1	1

[4]Not needed if the serial bus is unused.

Reset/Initialization Lines

Reset/Bus Initialization - RE*

The RE* signal is asserted by a module or the initialization and configuration process to initialize the bus interface logic of all modules on the bus to a predefined state. The duration of the asserted state on RE* is used to decide whether the signal represents live insertion, bus initialize, or system reset (see Table 3–3 and Table 3–4.)

Table 3–3 A Module Detecting Assertions of RE*

Time	Detecting RE*	Comments
0.001 to 1.999 ms	Live insertion	
2 to 30 ms	Bus initialize	Only the module initiating the bus init asserts RE*.
> 30 ms	Power up and system reset	Asserts after 30 ms release within 500 ms.
≥ 500 ms	Stuck RE*	If power up not occurring, indicates a stuck RE* line.

NOTE
Digital systems do not currently support live insertion.

Table 3–4 A Module Asserting RE*

Time	Asserting RE*	Comments
0.001 to 0.129 ms	Live insertion	
4 to 12 ms	Bus initialize	Only assert on allowed error cases.
100 ms to 200 ms	System reset	See the Power Up and System Reset sections of this chapter.

Geographical Address Lines

Geographical Address - GA[]*

These static lines encode the backplane slot number in which the module is located. This geographical address is used to uniquely establish the range of CSR addresses to which this module will respond.

These lines are asserted by each backplane slot and received by the module plugged into that slot.

Central Arbitration Lines

Central Arbiter Interface

On Profile B, arbitration proceeds as follows:

1. A Profile B module requests the use of the parallel bus using either RQ0* or RQ1*.
2. The central arbiter receives the request from the module (on RQ0* or RQ1*) and issues a grant using a fair arbitration scheme.
3. The module then becomes the master-elect.
4. ET* is released when the previous master is finished.
5. The master-elect then becomes the master and proceeds with its tenure and asserts ET* after AIf is released.

Central Arbitration Signal Names

Table 3–5 lists the signal names for the arbitration lines.

Table 3–5 Signal Names for Arbitration Lines

Prefix	Line
RQ0*	Request Low
RQ1*	Request High
GR*	Grant
PE*	Preempt

Types of Arbitration

Profile B supports two forms of arbitration:

- Central arbitration for the parallel protocol lines
- Distributed arbitration for arbitratration bus lines

Continued on next page

Request Low - RQ0*

Request Low - RQ0*:

- A module asserts RQ0* to request bus mastership at the lower of its two priorities (used for all transactions except Read Responses).
- Modules maintain RQ0* asserted until they become bus master and are performing their last transaction on the parallel bus.
- Modules may assert RQ0* even if they are asserting RQ1*.
- Modules should release RQ0* when they assert ET* but may hold RQ0* as late as when they release AS* on the last transaction of their tenure.

Request High - RQ1*

Request High - RQ1*:

- A module asserts RQ1* to request bus mastership at the higher of its two priorities (used for Read Responses only).
- Modules maintain RQ1* asserted until they become bus master and are performing their last transaction on the parallel bus.
- Modules may assert RQ1* even if they are asserting RQ0*. This allows a possible increase of their request priority. In this case, a module must maintain both RQ1* and RQ0* asserted until it becomes bus master and is performing its last transactions on the parallel bus.
- Modules should release RQ1* when they assert ET* but may hold RQ1* as late as when they release AS* on the last transaction of the tenure.

Continued on next page

Central Arbitration Lines, Continued

Grant - GR*

Grant - GR*:

- The central arbiter asserts GR* to indicate to a module that it is the master-elect (in response to RQ0* or RQ1*).
- The module then waits for ET* to release before beginning parallel bus transactions.
- The central arbiter maintains the assertion of GR* until both RQ0* and RQ1* from the master have been released.

Preempt - PE*

Preempt - PE*:

- The central arbiter asserts PE* to indicate to the master that a preemptive condition exists.
- The central arbiter releases PE* when it releases GR*.
- The pulse width of PE* may be shorter than that required by a module for recognition. Modules must allow for these conditions and operate correctly.
- When PE* is asserted within 100 ns of the release of AS*, it must be recognized by the master. The current bus master finishes its current transaction and releases its request to the central arbiter.
- A module can re-request the bus immediately after its grant is deasserted.

Arbitration (Distributed Arbitration) Bus Lines

Distributed Arbitration

In addition to central arbitration that is used to control access to the parallel protocol lines, Futurebus+ Profile B also supports distributed arbitration for the purposes of passing arbitration messages (such as the power failure messages).

In Digital systems, distributed arbitration is never used for purposes of gaining access to the parallel bus.

Arbitratration Bus Signal Names

Table 3–6 lists the signal names for the arbitration bus lines.

Table 3–6 Signal Names for the Arbitration Bus Lines

AP*, AQ*, AR*	Arbitration bus synchronization
AC[ ]*	Arbitration bus condition
AB[ ]*	Arbitration bus
ABP*	Arbitration bus parity

Arbitration Bus Synchronization - AP*, AQ*, AR*

Arbitration bus synchronization lines:

- Keep modules synchronized throughout a competition cycle.
- Operate in a 6-state sequence referred to as phase 0 through phase 5.

Arbitration Bus Condition - AC[]*

During phase 3 through phase 5, modules can assert AC0* and AC1* to signal an error.

Continued on next page

Arbitration (Distributed Arbitration) Bus Lines, Continued

Arbitration Bus - AB[ ]*

Modules compete for priority by applying their competition numbers (CN[ ]*) to AB[ ]* during phase 1. As modules recognize the presence of higher priority requests (evidenced by higher valued competition numbers), lower priority modules remove the less significant bits of their competition numbers. This process continues until the highest priority requester has been found.

Competition Details

The competition number presented on the bus is the "OR" of all of the individual competition numbers presented by the individual modules. For example, assume two modules are competing on the bus: Module A is asserting a competition number of 0x05 (using bits <2> and <0> while Module B is asserting a competition number of 0x03 (using bits <1> and <0>).

The "OR" of these two values is 0x07. Because this "OR" is greater in magnitude than either of the individual competition numbers, the situation could arise where both modules would temporarily remove their competition numbers from the bus, leaving zero. Both modules will then re-apply their competition numbers to the bus and the cycle would repeat. This scheme would not work.

To eliminate this effect, a module that seems to be losing the competition only removes a *portion* of its competition number from the bus. That portion consists of the bits between the LSB and the most significant bit that is asserted on the bus *but not by that module*.

Continued on next page

Competition Details (continued)

In the case of the example, Module B would recognize that bit <2> is asserted on the bus but not by Module B, and it would remove all bits of equal or lower significance. (In this example, Module B would remove all of its bits). Module A would also recognize that bit <2> is asserted on the bus but that it is one of the modules causing that assertion, therefore, it would allow bit <2> to remain. However, Module A would also recognize that bit <1> is asserted, but not by itself. Module A would therefore remove all bits of equal or lower significance.

Because both Module A and Module B stopped asserting bits <1> and <0>, the value on AB[ ]* would drop to 0x04. With bit <1> no longer asserted on the bus, Module A would again be free to assert bit <0> and the bus would finally stabilize at the correct winning value of 0x05, the value asserted by Module A.

Arbitration Bus Parity - ABP*

This line is used to provide the parity bit for AB[ ]*. While a competitor is asserting its full competition number on AB[ ]*, it shall also assert ABP* if the number of *ones* in its competition number is even.

Synchronization Lines

Overview

During a transaction, the synchronization lines coordinate the exchange of the following:

- Address
- Command
- Capability
- Status
- Data

Three-Line Protocol

The Futurebus+ address and data synchronization signals are arranged in a three-line configuration using a strobe from the master, a response from the selected slave (and any other nodes on the bus), and an inverted response from the slave (and any other nodes). This three-line configuration allows any node on the bus to stall the handshake at any point to meet the particular timing needs of that node.

The address synchronization lines operate in a straight-forward way with each line asserting and releasing once per address propagated. However, the Futurebus+ allows multiple data beats per bus transaction.

To minimize the number of signal transitions on the data synchronization lines, these lines operate in a fashion that allows a single transition (from asserted to released or released to asserted) per beat of data transferred. Only during the last data beat may two transitions be required (to enable the data synchronization lines to end up in the usual idle state).

Continued on next page

Signal Names

Table 3–7 lists the signal names for the synchronization lines.

Table 3–7 Signal Names for the Synchronization Lines

Prefix	Line
AS*	Address Sync
AK*	Address Acknowledge
AI*	Address Acknowledge Inverse
DS*	Data Sync
DK*	Data Acknowledge
DI*	Data Acknowledge Inverse

Address Synchronization Lines

Address Sync - AS*

AS* is asserted during the connection phase by the master to inform the system slaves that the address and command information is valid. Specifically, the master asserts AS* to indicate that AD[ ]*, BP[ ]*, CM[ ]*, CP*, and CA[ ]* have been activated with valid address, command, and capability information and that a transaction is being initiated.

AS* is released during the disconnection phase to indicate that the transaction on the bus is being terminated. The release of AS* indicates that AD[31..0]*, BP[3..0]*, CM[ ]*, and CP* have been activated with valid disconnection data.

Address Acknowledge - AK*

AK* is asserted by the master and all slaves when they detect AS* asserted. AK* is an acknowledge signal and is released by all modules during the disconnection phase.

Modules release AK* when they detect AS* released and they have asserted AI*.

The selected slave releases AK* when all of the following are true:

- The selected slave has detected AS* deasserted.
- The selected slave has asserted AI*.
- The selected slave has activated ST[ ]* with valid disconnection status.

Continued on next page

Address Synchronization Lines, Continued

Address Acknowledge Inverse - AI*

AI* is asserted by the master and all slaves when they detect AS* released. AI* is released by all modules during the connection phase.

All the slaves release AI* when all of the following are true:

- The slave has detected AS* asserted and has decoded the address from the signals on AD[ ]*.
- The slave has asserted AK*.
- If the slave is selected, it has activated the ST[ ]* lines with status and the CA[ ]* lines with desired capablity.

Data Synchronization Lines

Data Sync - DS*

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

The master asserts or releases DS* during the time AS* is asserted to indicate CM[ ]* and CP* have been activated with valid command information and the master requests the transfer of another word. During a write transaction, the transition of DS* also indicates that AD[ ]*, D[ ]*, and BP[ ]* have been activated with valid information.

If the master completes the data transfers for the current transaction and is asserting DS*, the master first releases AS* and then waits until it detects DK* released before releasing DS*.

Continued on next page

Data Acknowledge - DK*

DK* is an acknowledge signal asserted by the participating slave during the odd beats or odd packets and released by the participating slave during the even beats or even packets of the data phase. This is done to indicate to the master that status information and read data are valid or write data has been received.

In transactions, the participating slave asserts DK* when it detects DS* asserted. The slave releases DK* when all of the following are true:

- It detects the release of DS*.
- It has activated ST[7..1]* with valid status information.
- It has acquired information transmitted by the master on AD[ ]*, D[ ]*, and BP[ ]* if the transaction is a write.
- It has activated AD[ ]*, D[ ]*, BP[ ]* if the transaction is a connected read.
- It has asserted DI*.

Data Acknowledge Inverse - DI*

DI* is an acknowledge signal asserted by the participating slave during even beats or even packets and released by the slave during odd beats or odd packets of the data phase. This is done to indicate to the master that status information and read data are valid or write data has been received.

Continued on next page

Data Acknowledge Inverse - DI* (continued)

The participating slave asserts DI* when it detects the assertion of AS* in the connection phase. The slave also asserts DI* during even data beats when it detects DS* released. The slave releases DI* when all of the following are true:

- It detects the assertion of DS*.
- It has activated ST[7..1]* with valid status information.
- It has acquired information transmitted by the master on AD[ ]*, D[ ]*, and BP[ ]* if the transaction is a write.
- It has activated AD[ ]*, D[ ]* and BP[ ]* if the transaction is a connected read.
- It has asserted DK*.

End of Tenure Line

End of Tenure - ET*

The master asserts ET* when it detects AIf at logic zero and prior to the assertion of DS* during its first bus transaction, that is, when it first begins its bus tenure.

The current master releases ET* to indicate that its bus tenure has ended. The release of ET* is the indication that the current master has released all of the parallel bus lines and that the master-elect may begin a bus transaction.

If the master does not perform any transactions, it does not assert ET*.

Address/Data Lines

Address/Data Lines AD[31..0]*

The address/data lines AD[31..0] are multiplexed and bidirectional lines. The lines carry the following:

- Address from the master during the connection phase
- Data from the master in write transactions during the data phase
- Data from the slave in read transactions during the data phase
- Disconnection data during the disconnection phase

Accesses of 32-bit data are always transferred on AD[31..0]* independent of the actual width of a module.

During the *connection phase* of transactions, AD[31..5]* are activated with the address for the current transaction and AD[1..0]* are user-defined. Lines AD[4..2]* contain either address or user-defined information depending upon the particular data width (DW[ ]*) selected.

During the first data transfer of Read Partial or Write Partial transactions, AD[31..0]* carry the byte lane deselect signals from the master.

During the *data phase* of write transactions, AD[31..0]* are activated by the master with write data. During the data phase of read transactions these lines are activated by the slave with read data.

During the *disconnection phase*, these lines carry disconnection data.

Continued on next page

Read Response Connection

During the connection phase of a Read Response, the following events occur.

- AD[31..28]* are set to one.
- AD[27..12]* are activated with the global identification of the original requester.
- AD[11..6]* are reserved and must be released.
- AD[5..0]* are activated with the transaction identification of the original requester.

Read Response Disconnection

During the disconnection phase of Read Response transactions, the following events occur.

- AD[31..16]* are reserved and must be released.
- AD[15..8]* are activated with the arbitration priority PR[7..0]* of the requester.
- AD[8..3]* are reserved and must be released.
- AD2* contains the selected status SL* as asserted by the module at the ultimate destination of the request.
- Signal AD1* contains the conflict status. This bit is asserted by the module at the ultimate destination of the request when the module detects a potential deadlock condition. This requires the request transaction to be backed off all the way back to the originator. This is an end-to-end busy and the requester should not expect further responses.
- AD0* contains the transaction error status TE*. This bit is asserted if an error condition was detected anywhere along the transaction path of the request. The requester should not expect further responses.

Continued on next page

Address/Data Lines, Continued

Split Read Disconnection and Write Disconnection

During the disconnection phase of read transactions that have been split, write and write partial transactions, the following data shall be present on the AD[31:0]* lines.

- AD[31..16]* are activated with the global identification of the requester.
- AD[15..8]* are activated with the arbitration priority PR[7..0]* of the requester. These lines should be released.
- AD[7..6]* are reserved and must be released.
- AD[5..0]* are activated with a transaction identification.

Read Disconnection (Non-Split Reads)

During the disconnection phase of a non-split read, the disconnection data reflects the read data of the last beat. Parity may not be correct, therefore, no parity error should be flagged for this case.

Transferring Bytes

Bytes are transferred in their natural positions with no justification. During the first odd beat (the lane deselect beat) of Read Partial or Write Partial transactions, the following events occur.

- AD[31..0]* are defined as lane disable signals (L[ ]*) for the transfer occurring during the subsequent beats.
- The corresponding byte lane is not modified by the receiving slave if a lane deselect signal is asserted during Write Partial transactions.
- The corresponding byte lane is not evaluated by the master if a lane deselect signal is asserted by the master during the lane deselect beat of a Read Partial transaction.
- The lane deselect signals corresponding to lanes not enabled by the data width command are released.

Continued on next page

Address/Data Lines AD[63..32]*

These multiplexed address/data lines may carry an address during the connection phase and data during the data phase.

During the connection phase of a transaction with AW* asserted, AD[63..32]* are activated by the master with the most significant 32 bits of address for the current transaction.

During the connection phase of a transaction with AW* released, AD[63..32] are also released and are disregarded by all nodes on the bus.

During the data phase of write transactions with either DW1* or DW0* asserted, these lines are activated by the master with write data bits [63..32]. During the data phase of read transactions with either DW1* or DW0* asserted, these lines are activated by the slave with read data bits [63..32].

During the disconnection phase these lines are not defined and should be released.

Data Lines D[127..64]*

The data lines D[127..64] are bidirectional data lines that carry the data during a data transfer. These lines are not supported on Digital current implementations. Data is driven by the master during write transactions, and by the selected slave during read transactions.

During the connection phase and disconnection phase, D[127..64]* are not defined and must be released.

During the data phase of write transactions with DW1* asserted, D[127..64]* are activated with the write data bits [127..64] by the master. During the data phase of read transactions with DW1* asserted, D[127..64]* are activated with the read data bits [127..64] by the slave..

Byte Parity Lines

Byte Parity BP[]*

Each byte parity line carries the parity bit for one of the byte lanes. If the number of asserted signals in each byte lane is even, the byte parity signal is asserted.

Table 3–8 shows how the byte parity (BP[ ]*) lines correspond to the AD[ ]* lines and byte lanes.

Table 3–8 Byte Parity

Line	Carries Parity for:	Line	Carries Parity for:
BP0*	AD[7..0]*	BP8*	AD[71..64]*
BP1*	AD[15..8]*	BP9*	AD[79..72]*
BP2*	AD[23..16]*	BP10*	AD[87..80]*
BP3*	AD[31..24]*	BP11*	AD[95..88]*
BP4*	AD[39..32]*	BP12*	AD[103..96]*
BP5*	AD[47..40]*	BP13*	AD[111..104]*
BP6*	AD[55..48]*	BP14*	AD[119..112]*
BP7*	AD[63..56]*	BP15*	AD[127..120]*

Command Lines

Command Lines

The command lines CM[ ]* and CP* carry command information from the master to the slave for the transaction in progress.

Command Field - CM[ ]*

The master activates the command field with information that the slave uses to determine the type of transaction in progress or information about the transaction. These lines are uniquely defined in each of the transaction phases.

These signals are driven only by the master. Time periods when the signals on these lines are undefined are reserved for use by other standards and therefore, the lines should be released during these periods. Command signals change during the data phase of transactions only in multiple packet mode.

Command Field Table

Table 3–9 shows the command signals the master asserts during the various phases of a transaction. Undefined periods are denoted by a dash (–).

Continued on next page

Table 3–9 Command Fields

PHASE	CM7	CM6	CM5	CM4	CM3	CM2	CM1	CM0
Connection	AW*	DW1*	DW0*	WR*	TC3*	TC2*	TC1*	TC0*
Data Phase	DL2*	DL1*	DL0*	–	–	–	–	–
Address Only Disconnect	DL2*	DL1*	DL0*	–	–	–	–	–
Write Compelled Disconnect	DL2*	DL1*	DL0*	–	–	–	MR*	–
Split Request Disconnect	DL2*	DL1*	DL0*	–	–	–	MR*	–
Split Response Disconnect	DL2*	DL1*	DL0*	–	–	–	–	–

Key to Abbreviations

AW* is the address width indicator.
DW* is the data width indicator.
DL[ ]* is the data length indicator.
MR* is "MORE."
TC[ ]* is the transaction command.
WR* is "WRITE."
– is an unsupported command on Profile B modules.

Continued on next page

Address Width - CM7*

- During the connection phase, the CM7* line is defined as Address Width (AW*).
- The master asserts AW* to indicate that AD[63..32]* are activated with valid address information and that the slave(s) should decode a 64-bit address rather than a 32-bit address.
- Slaves incapable of decoding 64-bit addresses do not respond to transactions (shall not assert SL*) in which AW* is asserted.
- The master releases AW* to indicate that the slave(s) should decode the 32-bit address activated on AD[31..0]*. AD[63..32]* are to be ignored.
- During the data and disconnection phases, the CM7* line is defined as DL2*, bit 2 of the data transfer length.

Continued on next page

Data Transfer Length

The current master activates DL[ ]* to select one of the seven data transfer lengths or unrestricted transfer length, as shown in Table 3–10. This data transfer length refers to the number of transfers (data beats) that the master is requesting, not the number of bytes.

The maximum data length and data alignment for transactions are programmed at initialization by the configuration software using the LOGICAL_MODULE_CONTROL CSR.

Table 3–10 Data Length Field

DL2*	DL1*	DL0*	Data Transfer Length (Beats)
0	0	0	Unrestricted
0	0	1	1
0	1	0	2
0	1	1	4
1	0	0	8
1	0	1	16
1	1	0	32
1	1	1	64

Continued on next page

128 Bits - CM6*

During the connection phase, this line is defined as Data Width 1 (DW1*). The master asserts DW1* to indicate that the bus width for the transaction is 128 bits wide. The master releases DW1* to indicate that the transfer is either 32 bits or 64 bits.

During the data and disconnection phases, the CM6* line is defined as DL1*, bit 1 of the data transfer length.

64 Bits - CM5*

During the connection phase, this line is defined as Data Width 0 (DW0*). The master asserts DW0* to indicate that the bus width for the transaction is 64 bits wide, as shown in Table 3–11. The master releases DW0* to indicate that the transfer is either 32 bits or 128 bits.

During the data and disconnection phases, the CM5* line is defined as DL0*, bit 0 of the data transfer length.

Data Path Widths

Table 3–11 shows data path widths depending on the state of the DW1* and DW0*.

Table 3–11 Data Path Width

DW1*	DW0*	Path Width	Comments
0	0	32 bits	Supported on Digital office and computer room machines
0	1	64 bits	Supported on Digital office and computer room machines
1	0	128 bits	Not currently supported on Digital machines
1	1	Reserved	

Continued on next page

Transaction Command Selection

The master uses CM4*, CM3*, CM2*, CM1*, and CM0* to select one of the transaction commands shown in Table 3–12. On Profile B, all other encodings are reserved.

Table 3–12 Transaction Commmand Encoding

WR* (CM4*)	TC3* (CM3*)	TC2* (CM2*)	TC1* (CM1*)	TC0* (CM0*)	Transaction
0	0	0	0	0	Read Unlocked
1	0	0	0	0	Write Unlocked
1	0	0	0	0	Address Only Unlocked
0	0	0	1	0	Read Partial
1	0	0	1	0	Write Partial
1	0	1	0	1	Read Response

NOTE

The Read Response transaction has the WR* bit asserted because, at the time of the Read Response transaction, the node that is the current bus master is now writing response data back to a previous master that requested the original read function.

Master to Slave Data - CM4*

During the connection phase, this line is defined as write (WR*). The master asserts WR* to indicate that data is to be transferred from the master to the slave during the current transaction. The master releases WR* to indicate that data is to be transferred to the master from the slave.

Continued on next page

Transaction Command Line 3 - CM3*

During the connection phase, this line is defined as Transaction Command 3 (TC3*). The master activates TC[3..0]* and WR* to select one of the transaction commands shown in Table 3–12.

During the data and disconnection phases, this line is not defined and should be released.

Transaction Command Line 2 - CM2*

During the connection phase, this line is defined as Transaction Command 2 (TC2*). During the disconnection phase, this line is not defined and should be released.

Transaction Command Line 1 - CM1*

During the connection phase, this line is defined as Transaction Command 1 (TC1*).

During the data phase, this line should be released. During address only disconnection, compelled disconnection, and packet disconnection phases, MR* is asserted to indicate that a DMA block-copy operation is in progress. The block-copy transactions with CM1* (MR*) asserted are used to indicate that a read prefetch or writer buffering is allowed for this transaction ID. See Chapter 2 in this handbook for details on the use of MR*.

Transaction Command Line 0 - CM0*

During the connection phase, this line is defined as Transaction Command 0 (TC0*). During the disconnection and data phase, this signal should be released.

Command Parity Line

Command Lines Parity - CP*

This signal carries the parity for the command lines. If the number of asserted signals in CM[7..0]* is even, CP* is asserted.

Status Lines

Status Lines

The slave activates the status lines ST[ ]* to respond to the command of the master. The status lines provide the following information:

- Status of the slave
- Ability of the slave to participate in the transfer

In general, the master never asserts the status lines. The only exception to this is if a slave detects an error and asserts TE* and that slave then becomes the next master. The slave must continue to assert TE* until the end of the next connection phase, functioning as it would have had it not become the bus master.

Encoding Status Lines

Table 3–13 shows the encoding of the status lines. A reserved function (indicated by dash [-]) shall never be asserted by a Profile B module.

Table 3–13 Status Encoding

Phase	ST7*	ST6*	ST5*	ST4*	ST3*	ST2*	ST1*	ST0*
Connection	–	BE*	–	–	–	SL*	BS*	–
Data	–	BE*	–	–	–	SL*	ED*	TE*
Disconnection	–	BE*	–	–	–	SL*	BS* /ED*	TE*

Key to Abbreviations

BE* is a beat error.
BS* is busy.
ED* is end of data.
SL* is selected.
TE* is transaction error.
– is an unsupported status on Profile B modules.

Continued on next page

Connection Phase - ST[]* Status

These are a set of lines that are activated by the slave to respond to the command of the master. Signal lines ST[ ]* provide information about the status of the slave and the ability of the slave to participate in the transfer. During the connection phase, all modules are permitted to assert BE* if a parity error is detected on the AD[ ]* lines or the command lines.

Only the selected slave asserts SL* or BS* in the connection phase and activates the status lines during the data and disconnect phases. The master never drives the status lines, except in the case previously mentioned in the Status Lines section.

Wait - ST7*

This line is defined as Wait (WT*) and shall never be asserted by a Profile B module.

Beat Error - ST6*

This signal is defined as Beat Error (BE*).

During the connection phase:

- Any module asserts BE* if it detects an unsupported operation or parity error on CM[ ]*, AD[ ]* or CA[ ]*.
- Modules with 64-bit address capability also check parity on AD[63..32]* if AW* is asserted.
- Modules also assert BE* if the master requires the slave to respond with a capability that is lacking. If the master detects BE* asserted during the connection phase, then it proceeds to the disconnection phase.

During the compelled data phase:

- The slave asserts BE* if it detects an unsupported operation or parity error on CM[ ]*, AD[ ]*, D[ ]*, or CA[ ]*.
- If the appropriate data width (DW[ ]*) command bits were asserted, the slave also checks parity on AD[63..32]* and D[ ]*.

Continued on next page

Beat Error - ST6* (continued)

- The master can continue handshakes following the assertion of BE*, however, the data is ignored by both the master and slave.
- During the disconnection phase, modules that asserted BE* in the connection or data phase continue to assert BE* and must also assert TE*.

Intervention - ST5*

This line shall never be asserted by a Profile B module.

Transaction Flag - ST4*

This line shall never be asserted by a Profile B module.

Broadcast - ST3*

This line shall never be asserted by a Profile B module.

Selected - ST2*

This line is defined as Selected (SL*). Modules assert SL* during the connection phase when they recognize the address as one they should respond to and they do not detect any address or command parity errors. If the master does not detect SL* asserted at the end of the connection phase, it must proceed directly to the disconnection phase since none of the bus modules are responding.

Continued on next page

Busy - ST1*

During the connection phase, ST1* is defined as Busy (BS*). If the line is asserted during the connection phase, it is also defined as BS* during the disconnection phase. A selected slave asserts BS* to end the transaction without a data phase.

If BS* is not asserted during the connection phase, ST1* is defined as End of Data (ED*) during the data and disconnection phases. A slave asserts ED* during a compelled transaction to indicate that the current transaction must be terminated without further data transfers and the master is to proceed with the disconnection phase. To transfer more data, the master must initiate another transaction. During the disconnection phase, ED* indicates *end of data* status for the last compelled data transfer or the last requested packet.

The master may continue with handshakes following the assertion of ED*, however, the data is ignored by both the master and slave.

Continued on next page

Transaction Error - ST0*

This line is defined as Transaction Error (TE*). During the data or disconnection phase, modules assert TE* to report errors such as the following:

- Parity
- Packet overrun
- Packet underrun
- The master requesting packets beyond an end of data (ED*) status
- Disconnection parity

Modules maintain the assertion of TE* until the end of the connection phase of the next transaction. A module that asserts beat error (BE*) during the connection or data phase also asserts TE*. TE* is only sampled by the master in the disconnect phase, with the deassertion by the slave of AKf.

Capability Lines

Capability Lines - CA[ ]*

The module activates the capability lines CA[ ]* to communicate which mode of bus transaction it is able to perform. This activation provides the basic mechanism that allows modules with various transaction capabilities to coexist in the system. For example, a system may contain a module that responds to packet data transfers and a module that executes only compelled data transfers. In this case, the module that does not support packet data transfer causes the transaction to default to compelled data transfer mode.

Encoding Capability Lines

Table 3–14 shows the capability signal definitions for the various phases of a transaction.

Table 3–14 Capability Encoding

Phase	CA2*	CA1*	CA0*
Connection	SR*	CO*	TS*
Data	SR*	CO*	TS*
Disconnection	SR*	CO*	TS*

Key to Abbreviations

SR* is split response.
CO* is compelled.
TS* is transmit speed.

Continued on next page

Capability Lines, Continued

Split Response - CA2*

This line is defined as Split Response (SR*). A slave asserts SR* to require that the current transaction be split. The master then proceeds to the disconnection phase.

On Profile B modules, only Read Partial or Read Unlocked transactions can be split and only by the slave. A master never asserts CA2*. Write transactions and Read Response transactions are not permitted to be split.

Compelled - CA1*

This line is defined as Compelled (CO*). A module asserts CO* to require that the current transaction use the compelled protocols instead of the packet protocols.

Transmit Speed - CA0*

This line is defined as Transmit Speed (TS*). A module asserts TS* to request that the lower of the two software-selected packet-transmit speeds is used during the current transaction. If CO* is asserted, then TS* should be released. If a module supports only compelled data transfers, then CA0* need not be routed on the module.

Tag Lines

Tag Lines

Tag lines TG[ ]* are bidirectional lines that carry additional information related to the address/data lines (AD[63..0]* and D[127..64]*). The information represented by these lines is not specified in Profile B.

NOTE

The system implementer is responsible for ensuring that all modules within the system that use the Tag lines use them in a compatible way to prevent conflicts in usage.

Tag Parity Line

Tag Parity - TP*

This line carries the parity bit for TG[ ]*. If the number of asserted signals is even, TP* is asserted.

If no tag lines are active on this module (based on the CSR selection), this line is released. Tag parity checking is not required for Profile B compliance.

Arbitration

Gaining Tenure of the Bus

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

Digital systems use a simple central arbitration protocol. This protocol allows high speed operation and low request-to-grant latency.

Requesting the Parallel bus

Digital systems use a two-priority arbitration scheme that allows fairness to all modules on the bus. The higher priority request is used when requesting the parallel bus for a Read Response transaction. The lower priority request is used when requesting the parallel bus for all other transactions.

The Central Arbiter

To request bus mastership, a module asserts either RQ0* or RQ1*. The central arbiter samples the request signals from all modules and asserts GR* to the module that it determines has won the arbitration competition.

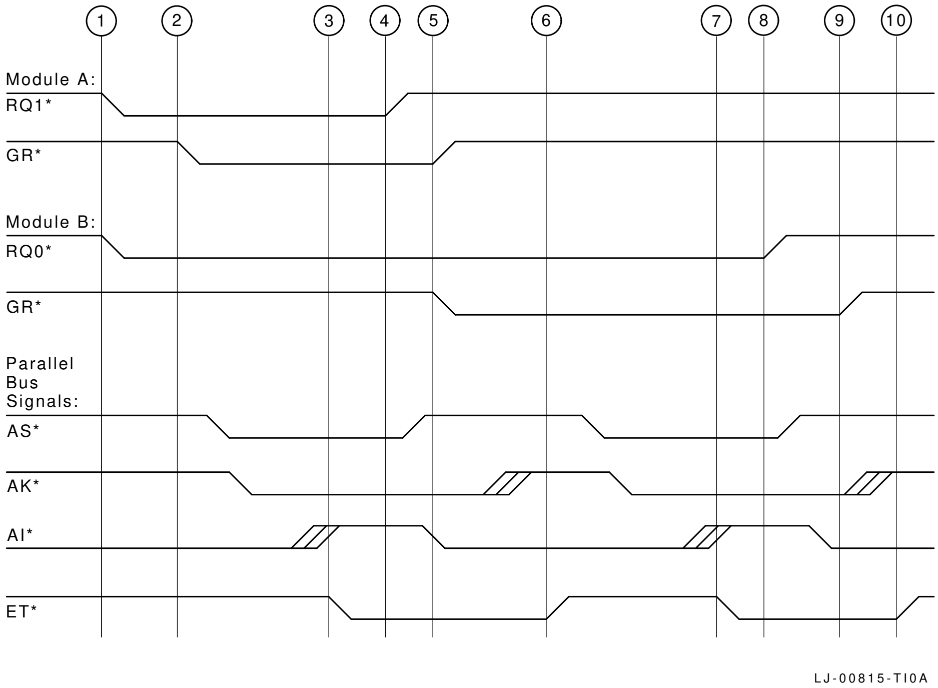
When the current master releases ET*, then the master-elect can start performing transactions on the parallel bus. The arbiter maintains GR* asserted until the current master has released both RQ0* and RQ1*.

Continued on next page

Arbitration without Preemption

Figure 3–2 shows the central arbitration competition procedures.

Figure 3–2 Two Competitor Central Arbitration



1. Modules A and B each assert their arbitration request. In this example, the request of Module A is of a higher priority than the request of Module B.
2. The central arbiter asserts GR* to Module A making Module A the master-elect. No other module is presently bus master (as evidenced by the released state of ET*), therefore, Module A immediately becomes the bus master and begins a bus transaction.

Continued on next page

Arbitration without Preemption (continued)

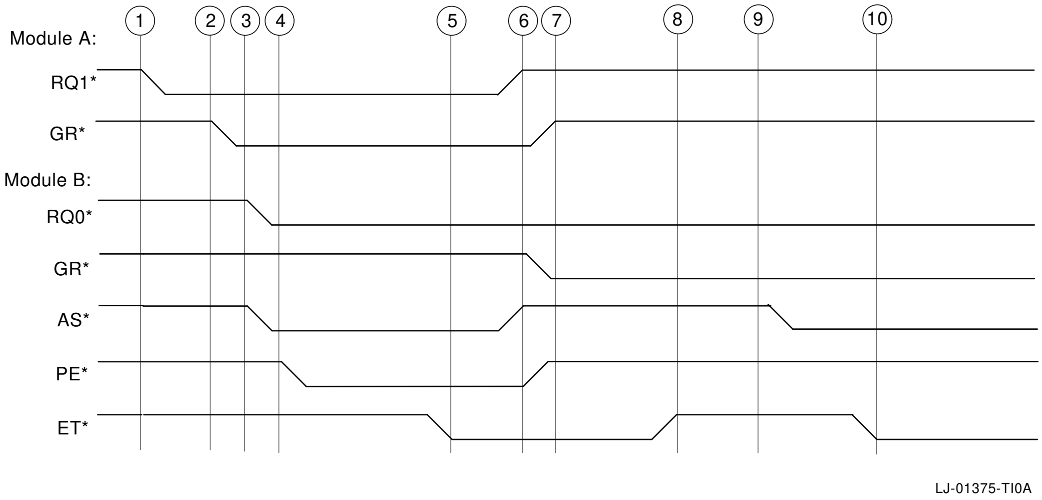
3. The current master, Module A, asserts ET* after beginning a transaction.
4. During its last transaction, Module A finishes its need for bus tenure and releases its request.
5. Upon detecting the request release by Module A, the central arbiter releases the grant to Module A, and asserts the grant to Module B. Module B then becomes the master-elect.
6. The current bus master, Module A, after releasing the parallel bus signals, releases ET* to indicate that any master-elect can become the new master.
7. After beginning its first transaction, the current master, Module B, asserts ET*.
8. During its last transaction, Module B finishes its need for bus tenure and releases its request.
9. Upon detecting that Module B has released its request, the central arbiter releases the grant to Module B. Since no requests are pending, the central arbiter issues no grant.
10. When the current master, Module B, has released the parallel bus signals, it releases ET*.

Continued on next page

Arbitration with Preemption

Figure 3–3 shows a preemptive case of central arbitration competition. In this example, the bus tenure of Module A is pre-empted.

Figure 3–3 Two Competitor Central Arbitration (Preemptive Case)



Continued on next page

Arbitration, Continued

Arbitration with Preemption (continued)

A preemptive case of central arbitration competition proceeds as follows. In this example, the bus tenure of Module A is preempted.

1. Module A asserts its arbitration request.
2. The central arbiter asserts GR* to Module A and Module A becomes the master-elect.
3. Module B asserts its arbitration request. For whatever reason, the request of Module B is of sufficient priority to preempt Module A.
4. The central arbiter asserts PE* to indicate a preemptive condition.
5. Module A asserts ET* after beginning a transaction.
6. Module A samples PE* prior to the deassertion of AS*, finishes its need for bus tenure, and releases its request.
7. Upon detecting the release of RQ1* by Module A, the central arbiter releases the grant to Module A, deasserts PE*, and asserts GR* to Module B. Module B then becomes the master-elect.
8. Module A, after releasing the parallel bus signals, releases ET* to indicate to any master-elect that the master-elect may now become bus master.
9. Module B begins its first bus transaction, asserting AS*.
10. Module B asserts ET* to indicate that it has assumed bus mastership.

Arbitration Bus

Overview

The Futurebus+ Arbitration Bus is used to:

- Perform distributed arbitration
- Send arbitrated messages to the central arbiter
- Send arbitrated messages amongst the various Futurebus+ modules

On current Profile B platforms provided by Digital, only the last use is supported, and the only message sent by Digital bus bridges is the required *Power Failure* message.

Structure

On Profile B, the Arbritration Bus can be viewed as an entirely separate bus from the Parallel Protocol Bus. It uses its own synchronization and data lines and its own unique protocol.

Continued on next page

Synchronization

Three wire-OR'd synchronization signals (AP*, AQ*, AR*) are used to keep all modules synchronized to the activities of the arbitration bus.

The activities of the bus are broken into six phases. These six phases may be viewed as three pairs of phases: The first phase of any pair is always begun by one module asserting one of the three synchronization lines. The second phase of any pair is always begun by all of the modules releasing another of the synchronization lines. When the last module releases the synchronization line, the second phase of the pair is actually entered. In this way, this protocol is fully compelled.

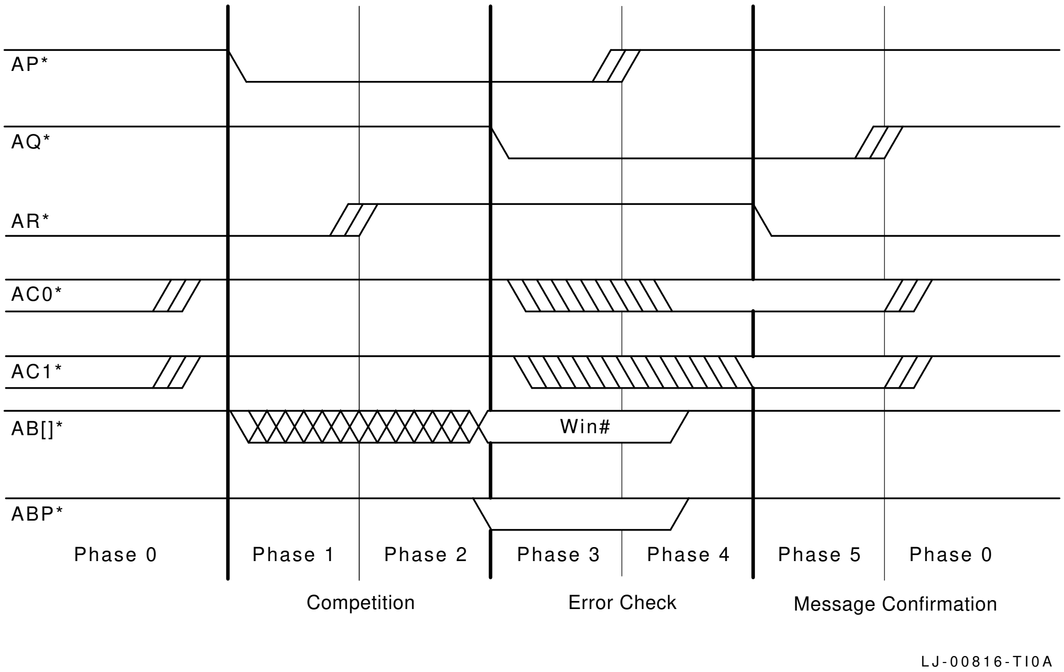
This also allows the protocol to advance states as rapidly as can be allowed by propagation delays and the time required to resolve metastabilities within the modules.

All modules that use the Arbritration Bus must remain synchronized to the activities on the bus and assert their synchronization signals in the specified fashion, even if they are not participating in the current arbitration.

Continued on next page

Figure 3–4 illustrates the various signals used by the Arbritration Bus. Figure 3–5 provides a state table for the three arbitration synchronization signals.

Figure 3–4 Arbitratration Bus Phases



Continued on next page

Figure 3–5 Arbitration Synchronization Signal State Table

Phase	AP*	AQ*	AR*	Phase-Pair Name
0	0	0	1	Idle
1	1	0	1	Competition
2	1	0	0	
3	1	1	0	Error-Check
4	0	1	0	
5	0	1	1	Message Confirmation
0	0	0	1	Idle

MKV-030000051-01-DGS

Continued on next page

Phase Overview

Phase 0, the idle phase, is the second half of the final phase pair. When no arbitration is ongoing, the arbritration bus rests in Phase 0.

Phase 1 is entered by any module that desires to begin a competition. The module indicates this by asserting AP*. All modules use this phase to decide whether or not to compete for this cycle of the arbitration bus. As modules resolve internal metastabilities and decide whether to compete, they release AR*. When the last module releases AR*, Phase 2 is entered and the competition begins. The various competing modules resolve their respective priorities as described in the Competition Details section earlier in this chapter.

Phase 3 is entered when a module decides that is has won the competition. The module indicates this by asserting AQ*. During this time, all modules latch and check the results of the competition. As each module decides that the results of the competition are valid, it releases AP*. When the last module releases AP*, Phase 4 is entered. In a distributed arbitration system, this phase is used to wait for the current bus master to end its tenure on the bus. In a central arbitration system such as Profile B, this phase is unused and all modules immediately assert AR* (which simulates the end of the tenure of the current bus master).

The assertion of AR* indicates the beginning of Phase 5. During this time, the message that was present on the Arbritration Bus is conveyed. As each module latches this status, the module releases AQ*. When the last module releases AQ*, Phase 0 is entered and the Arbritration Bus is again considered to be idle.

Continued on next page

Phase 0

Phase 0, the idle phase, is identified by AP* released, AQf at logic zero, and AR* asserted. This is the state of the arbitration bus synchronization lines when no competition cycle is in progress. Modules assert AP* to begin a competition. For example, the system bus bridge will assert AP* to begin issuing the Power Fail message.

Phase 1

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

After resolving any metastability, participating modules release AR*. No module releases AR* before it releases AC[1..0]*.

Phase 2

Phase 2, the competition phase, is identified by AP* asserted, AQ* released, and ARf at logic zero. Upon detecting the beginning of Phase 2, the system bus bridge begins timing the competition settling time.

Phase 3

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

During Phase 3, all modules latch the value of the winning competition number from the bus and check its parity. Competition losers also check that the winning competition number is greater than their own. Any modules detecting an error condition assert AC0*. Modules then release AP*.

Continued on next page

Phase 4

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

Phase 5

Phase 5 is identified by AP* released, AQ* asserted, and AR* asserted. In a central arbiter system, all modules immediately assert AR* to exit Phase 4. All modules save the state of AC[1..0]* during phase 5, to determine if an arbitrated message error has occurred or if this is the last competition cycle of a message. Once the modules latch this status, they release AQ*.

Power Fail - Arbitrated Message

The only arbitrated message transmitted by the system bus bridge in Digital systems is Power Fail. Power Fail is an emergency message that is decoded as shown in Table 3–15. Any message in the range of 0x60 through 0x7F is considered to be a Power Fail Message.

Table 3–15 Power Fail Decodings

Bit	Value	Bit	Value
CN7[1]	1	CN3	x
CN6	1	CN2	x
CN5	1	CN1	x
CN4	x	CN0	x

1 = Not part of the competition number
x = Do not care

Continued on next page

Power Fail - Arbitrated Message (continued)

Central Arbiter messages and general arbitrated messages will be ignored by current Digital system bus bridges and the central arbitration logic.

Minimum Power Fail Implementation

A Profile B module that needs to receive the Power Fail message may implement a minimum power fail receive circuit. The four wires that are needed are AQ*, AB[7..5], AB[6], and AB[5]*. If AB[7], AB[6] and AB[5]* are asserted when AQ* asserts, then the module can initiate a power fail sequence. This minimum implementation will not support the use of distributed arbitration. Distributed arbitration messages will be construed as Power Fail messages.

If a module does not need to decode Power Fail, it does not need to connect to the arbitration bus.

Refer to the IEEE 896.1 standard if a modules needs to implement the full arbitrated message scheme and/or distributed arbitration.

Digital systems do not support arbitrated messages or distributed arbitration messages (except for the transmission of Power Fail).

Transactions

Conducting a Transaction

After a module has received a grant from the central arbiter, it uses the parallel bus to conduct transactions. A transaction consists of a request followed by a response. There are two ways to conduct a transaction:

1. A connected transaction is used to conduct the request of a module and the response in a single bus transaction.
2. A split transaction is used to split the request and response phases into separate bus tenures and transactions. The module servicing the request responds by becoming a master, addressing the requester, and transmitting a response to the requester.

Bus Transaction Phases

A master conducts transactions during its bus tenure. Each transaction is comprised of up to three phases:

- Connection
- Data (optional)
- Disconnection

All transactions are required to have a connection and a disconnection phase.

Connection Phase

During the connection phase, a master establishes a connection between the master and the desired slave. The master drives the AD[ ]*, CA[2..0]* and CM[7..0]* lines and then asserts AS*.

The master and slave resolve any differences in their capabilities during the connection phase. The master specifies the type of transaction it wishes to perform. If the slave cannot perform the specified transaction, it either asserts BE*, or if possible, converts the transaction into one it is capable of performing. The end of the connection phase is signaled by the release of AIf.

Continued on next page

Data Transfer Phase

When the data phase occurs, data are transferred between the master and the selected slave. This phase begins when the master asserts DS* and ends when the master enters the disconnection phase.

The master transfers data with the selected slave during this phase. During the data transfer phase, the master conducts data transfers using the method agreed upon during the connection phase, which is either compelled mode or packet mode.

The data phase is not always a part of every transaction.

Address Only Transactions

The data transfer phase is not included in some types of transactions. Transactions without a data transfer phase are referred to as address only transactions. This type of transaction is used to implement split transactions. Address only transactions may also occur if an error is detected in the connection phase.

Data Beat

A data beat begins with the transition of DS* made by a master. This is followed by the release of an acknowledge line (DI* or DK*) by the slave. Command and data information is sent from the master to the slave during the first half of a beat, and status and data information may be returned to the master from the slave during the second half.

Odd/Even Data Beats

Data beats in which the master asserts DS* are referred to as odd beats. Conversely, data beats in which the master releases DS* are referred to as even beats. Protocols in which both edges of a synchronization signal are used are referred to as two-edge handshakes.

Continued on next page

Compelled Transactions

The compelled protocol is a technology-independent transfer mechanism. This type of handshake is referred to as compelled because the slave is compelled to provide a response before the master proceeds.

The protocol to be used is negotiated during the connection phase of the transaction. Any participant may force the transfer to be performed with the compelled protocol.

Compelled Data Transaction

A compelled data transaction consists of an address transfer followed by a block of one or more data phases. This proceeds to access one or more contiguous addresses beginning with the address specified in the address transfer. Each data transfer is controlled by a fully compelled handshake between the master and slave. On current Digital systems, transfers to and from system memory must be compelled. Transfers to and from system memory on current Digital system bus bridges must be compelled.

Types of Transactions

Futurebus+ Profile B modules shall support the following compelled transactions:

- Read Partial
- Read Response
- Read Unlocked
- Write Unlocked
- Write Partial
- Address Only Unlocked

Continued on next page

Packet Data Transfer

A packet data transfer consisting of an address transfer followed by a fixed length block of data, transfers from a set of contiguous addresses. This process begins with the address in the address transfer. Data transfers within a packet use a noncompelled protocol. Each complete packet transfer is controlled by a fully compelled handshake between the master and the slave. Current Digital implementations do not support packet mode.

Disconnection Phase

The disconnection phase is the phase in which the master terminates the transaction and disconnects from the slave. Disconnection information is transmitted from the master to the slave on the AD[ ]* and CM[ ]* lines during the beginning of the disconnection phase. Status is transmitted back to the master from the slave at the end of the disconnection phase, with the deassertion of AKf. In compelled transactions, data transfers may be of any length within the constraints of the COMPELLED_ DATA_LENGTH field of the LOGICAL_MODULE_CONTROL CSR.

Packet Transactions

Packet protocol is a technology-dependent, noncompelled transfer mechanism at the individual transfer level and a compelled protocol at the packet level. The packet protocol is optimized for high speed block transfers and does not include many of the features available in the compelled protocols.

Module Transfer Rates

The participants in a packet transaction must all operate at the same transfer rate. All of the modules capable of operating using packet mode perform a system-wide negotiation during initialization. This is done to configure the two standard transfer rates for that system. Each of these modules provide a register from which other modules may read its supported transfer rates (see IEEE 896.2, Bus Management chapter, for details).

Continued on next page

High and Low Transfer Speeds

The system initialization and configuration process reads the supported transfer rates from all the other modules. The low transmit speed is set equal to the lowest speed of all the packet-capable modules on the bus. The high transmit speed is set equal to the highest speed at which the majority of the modules can operate. The determination of which of the two transmit speeds will be used during a particular transaction is negotiated during the connection phase of the transaction. Any participating module may force the transfer to be performed using the lower speed.

Packet Lengths

Packet lengths result in multiple data transfers occurring after a single data handshake in the packet mode. Participating modules must know in advance how many transfers to perform. Each module capable of packet operation provides a register that specifies its supported packet lengths (see IEEE 896.2, Bus Management chapter, for details). Only 2, 4, 8, 16, 32, and 64 "word" transfer length packets are permitted. (In this case, "word" means the current data width of the bus, for example, 64 bits for a D64 transaction.) The master specifies one of these standard packet lengths during the data phase.

Split Transactions

Only Read transactions are allowed to be converted to split transactions. Split transactions allow modules with long data latency (for example, a long access time) to use the bus resources in a more efficient way. A complete split transaction requires two bus transactions.

Split transactions require additional information beyond that necessary for connected transactions. For this purpose, facilities are provided to transfer the global identification and original priority of the requester, and the status of the responder in the disconnection phase.

All potential bus masters must support split response capability.

Continued on next page

Splitting a Read Transaction

If a read transaction is initiated and the participating slave requires the transaction be split, the connected transaction is converted to an address only transaction which then requests the desired data. Sometime after the master (the requester) which requested the data ends its bus tenure, the module (the responder) with the requested data acquires the bus and transmits the data to the requester.

Transactions Terminating with Busy

If an addressed module is unable to respond with a particular resource at the time it is requested to do so, the module notifies the master that it is Busy. For example, this may happen if the resource is currently being accessed by another port. Transactions terminated with busy do not change the system state. A master provides a forward progress mechanism through the use of the Busy Retry registers described in IEEE 896.2, Chapter 3, and in Chapter 2 of this handbook.

Transactions Resulting in Starvation

Forward progress is guaranteed in a system that does not exhibit deadlock or starvation. Starvation occurs when one or more modules perform no useful work for an indefinite period of time. Deadlock occurs when all actors are waiting on actions that can only be performed by those already waiting. Starvation may result from livelock, a situation in which some modules acquire and release resources in a way that none is making progress.

While it is possible to build a Futurebus+ system guaranteeing no deadlock, designs using multiple buses may be vulnerable to livelock, and should be designed very carefully to minimize the probabilities of events that might result in such behavior. Most situations that could result in livelock result from the use of Busy.

Read Unlocked Transactions

Definition

The Read Unlocked transaction is used to transfer data from the slave to the master.

The slave may convert this transaction into a split transaction. In this case, the Read Unlocked transaction may be referred to as a Read Unlocked request.

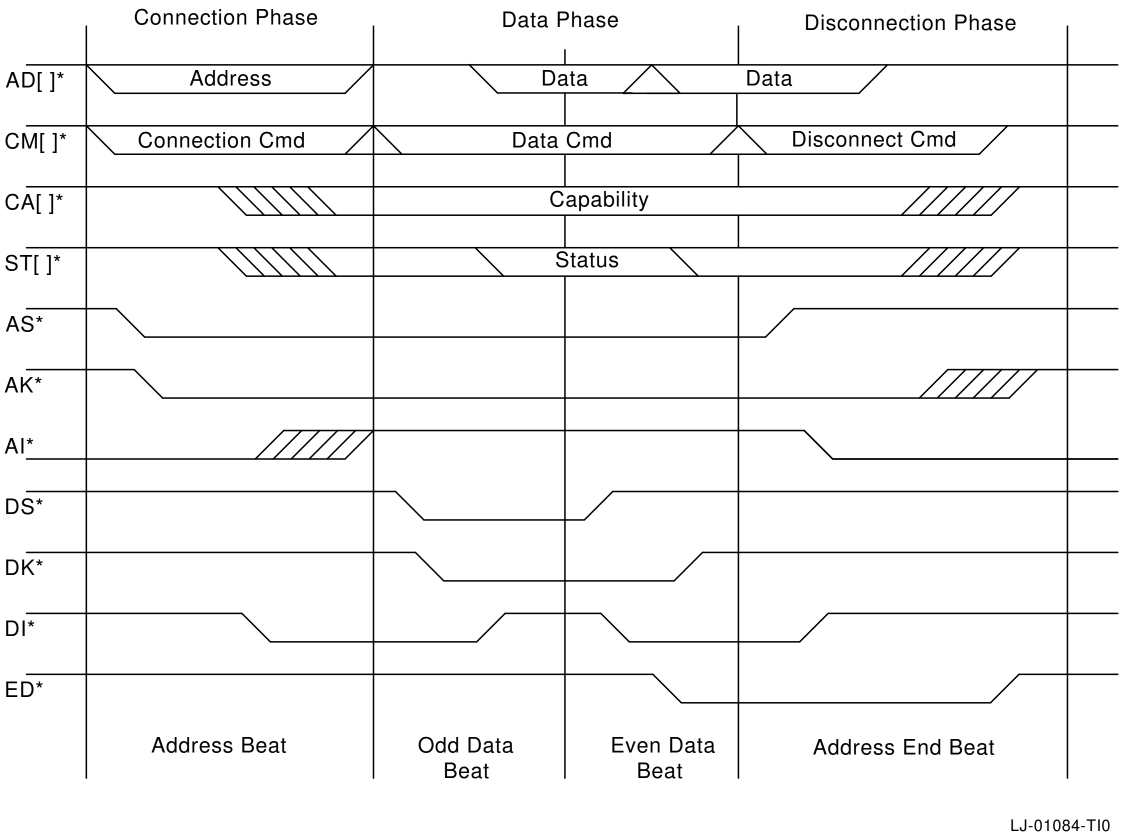
Sequence of Events

The connected, compelled Read Unlocked transaction begins with a connection phase. The master then enters the data phase and requests one or more beats of data from the selected slave. The master then enters the disconnection phase.

Figure 3–6 and Table 3–16 illustrate and detail the steps of a Read Unlocked transaction.

Continued on next page

Figure 3–6 Compelled Read Transaction



Continued on next page

Table 3–16 Steps of a Read Unlocked Transaction

Step	Action	Description
1	Activate	Master activates the AD[ ]* and BP[ ]* lines with the address and address parity, the CM[ ]* and CP* lines with the appropriate connection command bits and command parity, and the CA[ ]* lines with capability information.
2	Assert	The master then asserts AS* to mark the validity of these lines.
3	Evaluate	When the slave detects AS* asserted, it asserts AK* and evaluates the address and the command information.
4	Notify	If the address is one to which it responds, the slave becomes selected and asserts SL* as shown to notify the master that it is responding to the transaction. The slave also asserts any of the other status lines that are applicable. DI* is also asserted at this time in preparation for the odd data beat to follow.
5	Release	The slave releases AI*
6	Detect	When the master detects AIf at a logic zero, it evaluates the ST[ ]* (status) and CA[ ]* (capability) lines.
7	Activate	Master activates CM[ ]* and CP* with a data phase command and command parity, and releases AD[ ]*.
8	Assert	Master asserts DS* to request data from the slave.
9	Detect	Slave detects DS* asserted.
10	Evaluate	Slave evaluates CM[ ]* and asserts DK*.
11	Activate	When requested data is available, the slave activates AD[ ]* and BP[ ]* with requested data.
12	Notify	Slave releases DI* to notify the master that the data is valid.
13	Detect	Master detects DI* released.
14	Capture	Master captures the data and releases DS* to request data from the next address.
15	Detect	Slave detects the release of DS* and asserts DI*.
16	Activate	When requested data is available, the slave activates AD[ ]* and BP[ ]* with requested data.

Continued on next page

Table 3–16 (Continued) Steps of a Read Unlocked Transaction

Step	Action	Description
17	Assert	If the transfer reaches an address beyond which the slave cannot continue, the slave asserts ED* to indicate that it can support no further data transfers past the current one.
18	Release	Slave releases DK*.
19	Detect	Master detects release of DK* and assertion of ED*.
20	Capture	Master captures the data on AD[ ]* and proceeds to disconnect phase as in the address only transaction.

NOTE

In the interest of increased bus utilization efficiency, Digital recommends that the Read Partial transaction not be used. Instead, a Read Unlocked transaction should be used and the master should internally mask the returned data.

Read Partial Transaction

Definition

The Read Partial transaction is used to transfer data from the slave to the master. This transaction is very similar to the Read Unlocked transaction except that the master specifies that it will only evaluate some of the bytes transferred to it by the slave.

The slave may convert this transaction into a split transaction. In this case, the Read Partial transaction may be referred to as a Read Partial request.

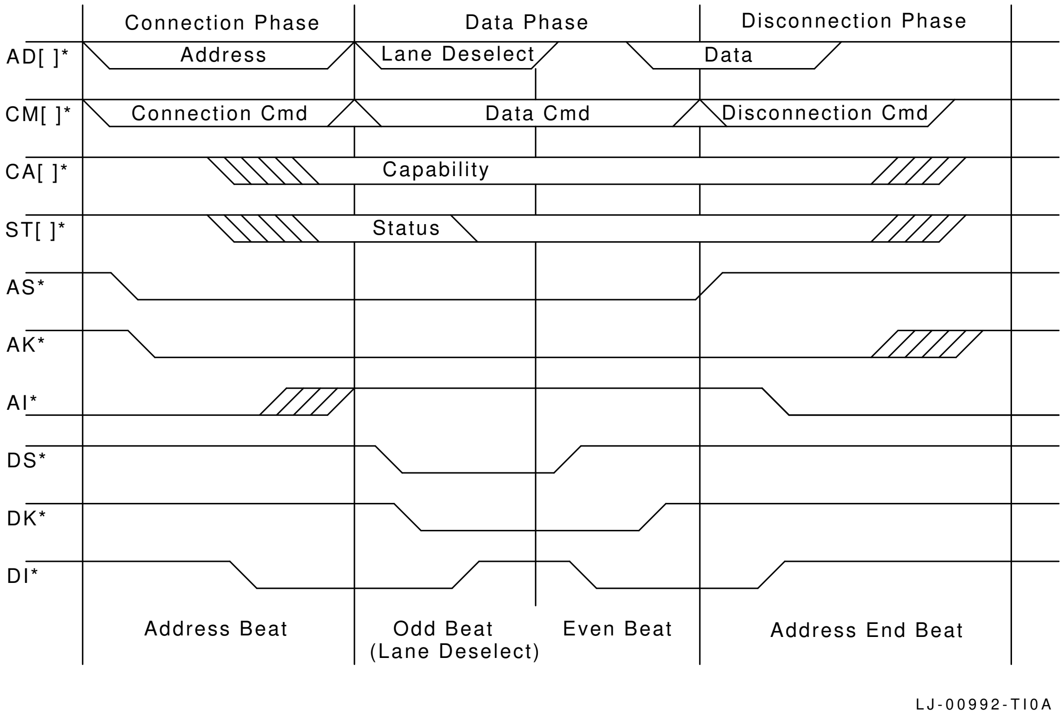
Sequence of Events

The connected, compelled Read Partial transaction begins with a connection phase. The master then enters the data phase and uses the first data beat to transfer a byte lane deselect mask to the selected slave. The master then uses the next data beat(s) to request the actual data. In a typical Read Partial transaction, only one beat of data is transferred to the master. The master then enters the disconnection phase.

Figure 3–7 illustrates a Read Partial transaction. Table 3–17 details the additional steps that a Read Partial transaction must perform as compared to a Read Unlocked transaction.

Continued on next page

Figure 3–7 Compelled Read Partial Transaction



Continued on next page

Read Partial Transaction, Continued

Table 3–17 Steps of a Read Partial Transaction

Step	Action	Description
1	Transfer	On the first odd beat, the master transfers lane deselect information on AD[31..0]* and the appropriate parity information on BP[ ]*.
2	Locate	On the even beat, the slave sources the requested data.
3	Response	Upon receiving the data, the master ignores byte lanes if the corresponding lane deselect signal was asserted.

NOTE

In the interest of increased bus utilization efficiency, Digital recommends that the Read Partial transaction not be used. Instead, a Read Unlocked transaction should be used and the master should internally mask the returned data.

Split Read Transactions

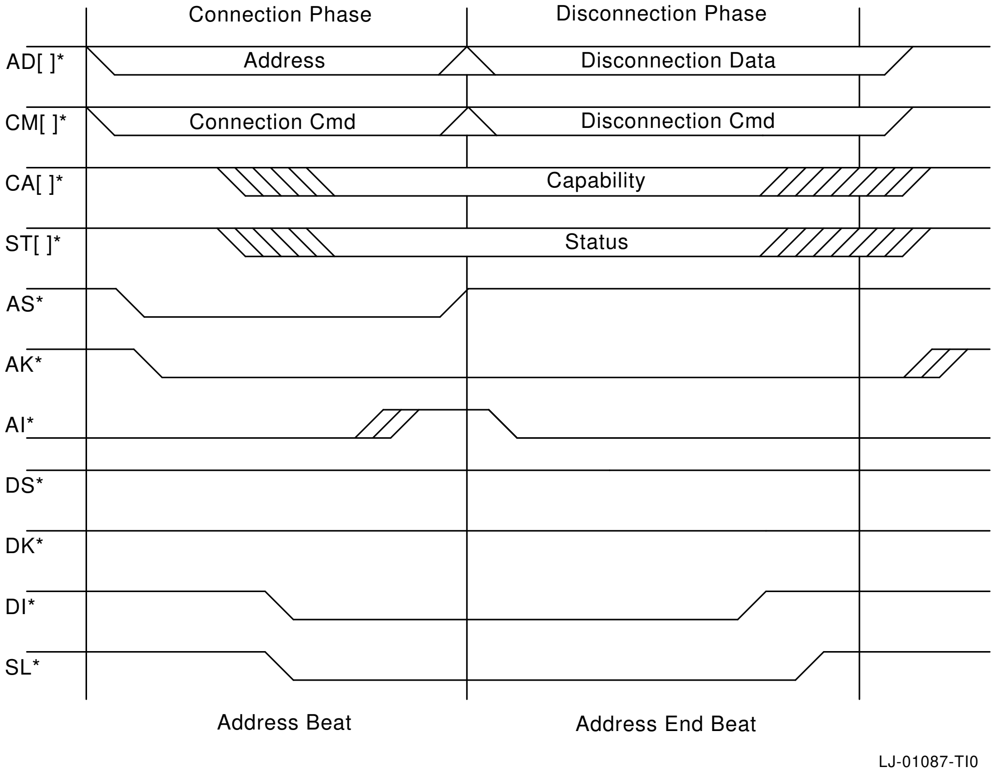
Definition

If the slave is unable to respond with read data, it may split any Read Unlocked or Read Partial transaction by asserting SR* during the connection phase. The master then enters the disconnection phase and passes information to the slave regarding the format of the eventual Read Response.

Sequence of Events

A split Read Unlocked transaction consists of only a connection and disconnection phase. Figure 3–8 shows an Address Only portion of a split Read Unlocked. Figure 3–8 and Table 3–18 illustrate and detail the steps of this transaction for the slave and master.

Figure 3–8 Address-Only Portion of a Split Read Unlocked Transaction



Continued on next page

Table 3–18 Steps of a Split Read Unlocked Transaction

Step	Action	Description
1	Activate	Master activates the AD[ ]* and BP[ ]* lines with the address and address parity, the CM[ ]* and CP* lines with the appropriate connection command bits and command parity, and the CA[ ]* lines with capability information.
2	Assert	The master then asserts AS* to mark the validity of these lines.
3	Evaluate	When the slave detects AS* asserted, it asserts AK* and evaluates the address and the command information.
4	Notify	The selected slave asserts SL* as shown to notify the master that it is responding to the transaction. The slave also asserts any of the other status lines that are applicable, including SR*. DI* is also asserted at this time in preparation for the odd data beat to follow.
5	Release	The slave releases AI*.
6	Detect	When the master detects AIf at a logic zero, it evaluates the status and capability lines.
7	Disconnect	When the master detects SR*, it proceeds to the disconnect phase. The master activates AD[ ]* and BP[ ]* with disconnection data, CM[ ]* and CP* with the disconnection command and command parity.
8	Validate	The master then releases AS* to mark the validity of AD[ ]*, BP[ ]*, CM[ ]*, and CP*, and to indicate to the participating slave that the transaction is being terminated.
9	Evaluate	The slave, upon detecting the release of AS*, evaluates the information on AD[ ]*, BP[ ]*, CM[ ]*, and CP* and then asserts AI*, activates ST[ ]*, and releases AK* and DI*.
10	Transaction End	When the master and slave detect AKf at logic zero, they release AD[ ]*, BP[ ]*, CM[ ]*, CP*, CA[ ]*, and ST[ ]* and the transaction ends.

Read Response Transaction

Definition

A Read Response transaction is the response to a previous request. During the connection phase the identification and status of the original requester are transmitted. The requester uses this information to know that this response and the forthcoming data correspond to its own request. During the data phase, the data is sent to the original requester.

Slaves must not (further) split Read Response transactions.

Sequence of Events

The Read Response begins with a connection phase. The address and data lines in the connection phase contain the following information necessary to reconnect to the original requester.

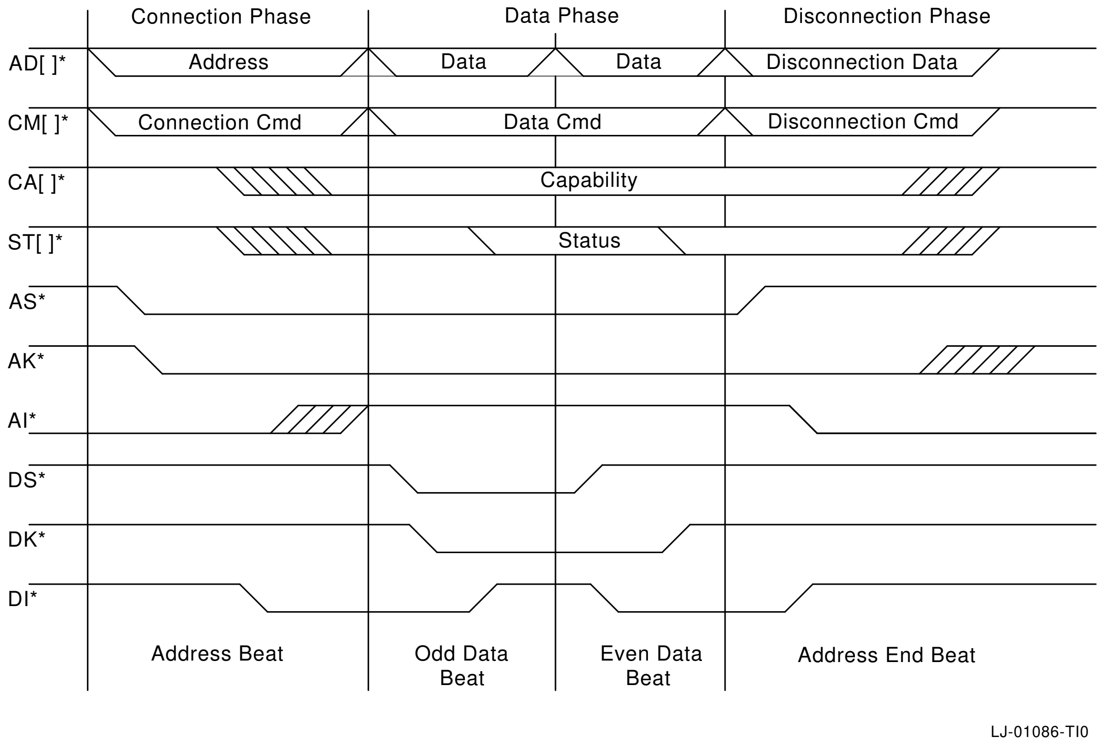
- AD[31..28]* are set to one.
- AD[27..12]* are activated with the identification of the original requester.
- AD[11..6]* are reserved.
- AD[5..0]* are activated with the transaction identification, an arbitrary value supplied by the original requester.

The master then enters the data phase and transfers one or more beats of data to the selected slave (the original requester). The master then enters the disconnection phase.

Figure 3–9 and Table 3–19 illustrate and detail the steps within this transaction for the slave and master.

Continued on next page

Figure 3–9 Compelled Read Response Transaction



Continued on next page

Table 3–19 Steps of a Compelled Read Response Transaction

Step	Action	Description
1	Activate	Master activates the AD[ ]* and BP[ ]*lines with the connection information (previously described) and the connection information parity, the CM[ ]* and CP* lines with the appropriate connection command bits and command parity, and the CA[ ]* lines with capability information.
2	Assert	The master then asserts AS* to mark the validity of these lines.
3	Evaluate	When the slave detects AS* asserted, it asserts AK* and evaluates the address and the command information.
4	Notify	If the reconnection information indicates that this is the slave to be selected, the slave then asserts SL* as shown to notify the master that it is responding to the transaction. The slave also asserts any of the other status lines that are applicable DI* is also asserted at this time in preparation for the odd data beat to follow.
5	Release	The slave releases AI*.
6	Detect	When the master detects AIf at a logic zero, it evaluates the ST[ ]* (status) and CA[ ]* (capability) lines.
7	Activate	Master activates CM[ ]* and CP* with a data phase command and command parity and AD[ ]* and BP[ ]* with the data and data parity.
8	Assert	Master asserts DS* and notifies the slave that the CM[ ]*, CP*, AD[ ]*, and BP[ ]* lines are valid.
9	Detect	Selected slave detects DS* asserted.
10	Assert	Selected slave asserts DK*.
11	Capture	Selected slave asserts DK* and captures the data on the AD[ ]* lines.
12	Release	Slave releases DI* and notifies the master that it has captured the data and that ST[ ]* are valid.
13	Detect	Master detects DI* released.
14	Capture	Master captures the slave status.
15	Activate	Master activates AD[ ]* and BP[ ]* with the next read data and data parity and releases DS*.
16	Detect	Selected slave detects DS* released.

Continued on next page

Table 3–19 (Continued) Steps of a Compelled Read Response Transaction

Step	Action	Description
17	Assert	Selected slave asserts DI*.
18	Capture	Selected slave captures data on the AD[ ]* lines.
19	Release	Slave releases DK* to notify the master that it has captured the data and that the ST[ ]* lines are valid.
20	Detect	Master detects DK* released.
21	Capture	Master captures slave status and proceeds to the disconnection phase.

Write Unlocked Transaction

Definition

The Write Unlocked transaction is used to transfer data to the slave from the master.

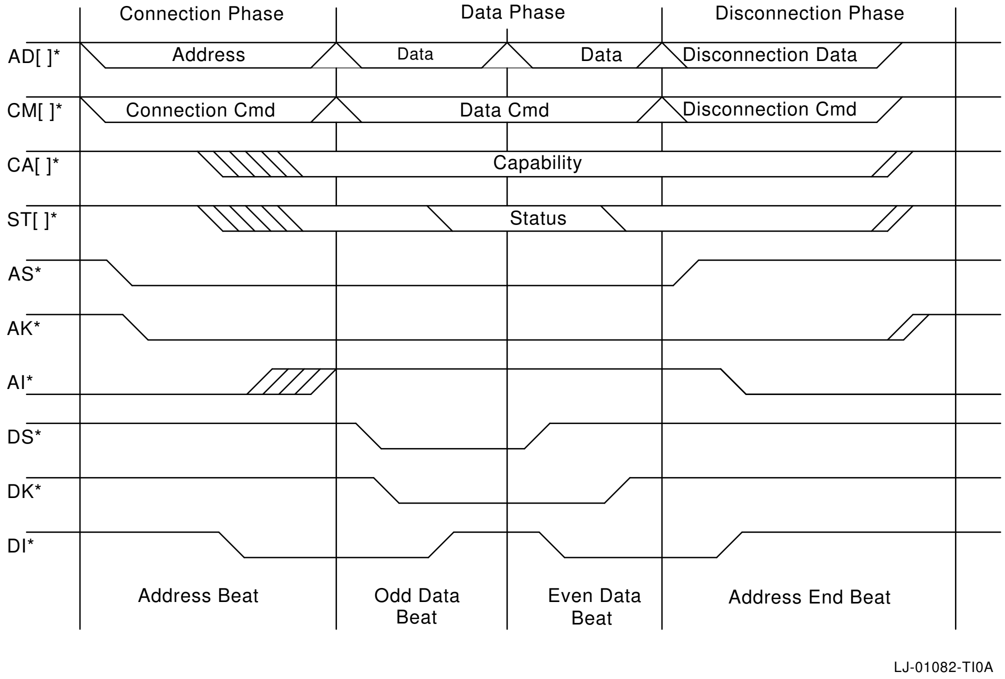
Slaves must not split Write Unlocked transactions.

Sequence of Events

The connected, Compelled Write Unlocked transaction begins with a connection phase. The master then enters the data phase and transfers data to the selected slave. Finally, the master enters the disconnection phase.

Figure 3–10 and Table 3–20 illustrate and detail the steps of a Write Unlocked transaction.

Figure 3–10 Compelled Write Unlocked Transaction



Continued on next page

Table 3–20 Steps of a Compelled Write Unlocked Transaction

Step	Action	Description
1	Activate	Master activates the AD[ ]* and BP[ ]*lines with the address and address parity, the CM[ ]* and CP* lines with the appropriate connection command bits and command parity, and the CA[ ]* lines with capability information.
2	Assert	The master then asserts AS* to mark the validity of these lines.
3	Evaluate	When the slave detects AS* asserted, it asserts AK* and evaluates the address and the command information.
4	Notify	If the address is one to which it should respond as a selected slave, the selected slave asserts SL* as shown to notify the master that it is responding to the transaction. The slave also asserts any of the other status lines that are applicable.
5	Release	The slave releases AI*.
6	Detect	When the master detects AIf at a logic zero, it evaluates the ST[ ]* (status) and CA[ ]* (capability) lines.
7	Activate	Master activates CM[ ]* and CP* with a data phase command and command parity and AD[ ]* and BP[ ]* with the write data and data parity.
8	Assert	Master asserts DS*, notifying the slave that CM[ ]* and AD[ ]* are valid. DI* is also asserted at this time in preparation for the odd data beat to follow.
9	Detect	Selected slave detects DS*.
10	Assert	Selected slave asserts DK*.
11	Capture	Selected slave captures the data on AD[ ]*.
12	Release	Slave releases DI* to notify master it has captured the data, and ST[ ]* are valid.
13	Detect	Master detects DI*.
14	Capture	Master captures the slave status when released.
15	Activate	Master activates AD[ ]* and BP[ ]* with the next write data and data parity and releases DS*.
16	Detect	Selected slave detects DS* released.

Continued on next page

Write Unlocked Transaction, Continued

Table 3–20 (Continued) Steps of a Compelled Write Unlocked Transaction

Step	Action	Description
17	Assert	Selected slave asserts DI*.
18	Capture	Selected slave captures the data on the AD[ ]* lines.
19	Notify	Slave releases DK* to notify the master that it has captured the data and that the ST[ ]* lines are valid.
20	Detect	Master detects DK* released.
21	Capture	Master captures the slave status and proceeds to the disconnection phase.

Write Partial Transaction

Definition

The Write Partial transaction is used to transfer data from the master to the slave. This transaction is very similar to the Write Unlocked transaction except that the master specifies that the slave should only evaluate some of the bytes transferred to it by the master.

Slaves must not split Write Partial transactions.

Sequence of Events

The connected, compelled Write Partial transaction begins with a connection phase. The master then enters the data phase and uses the first data beat to transfer a byte lane deselect mask to the selected slave. The master then uses the next data beat(s) to transfer the actual data. In a typical Write Partial transaction, only one beat of data is transferred to the slave. The master then enters the disconnection phase.

Figure 3–11 illustrates a Write Partial transaction. Table 3–21 details the additional steps that a Write Partial transaction must perform as compared to a Write Unlocked transaction.

Continued on next page

Figure 3–11 Compelled Write Partial Transaction

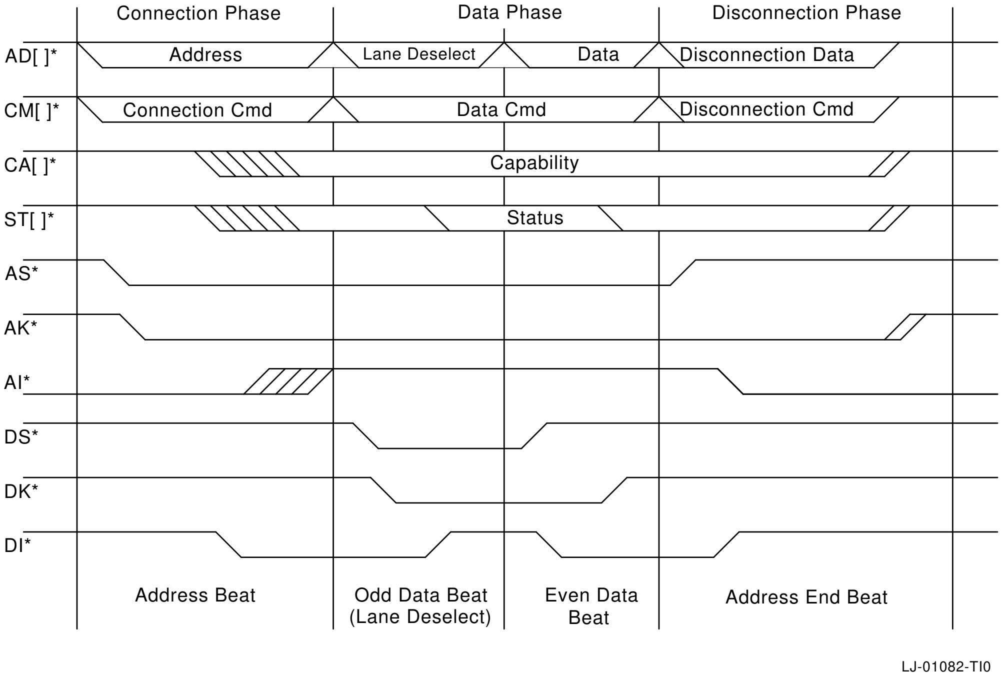


Table 3–21 Steps of a Compelled Write Partial Transaction

Step	Action	Description
1	Transfer	On the first add data beat, the master transfers lane deselect information on AD[31..0]* and the appropriate parity information on BP[ ]*.
2	Transfer	On the first even data beat, the master transfers the data to be written.
3	Mask	The slave evaluates only the data in those byte lanes that were not deselected by the lane deselect information transferred earlier.

Unsupported Futurebus + Protocol

Unsupported Transactions

The following transactions are not supported under Profile B. If a Profile B module is targeted by one of these transactions, it shall respond by asserting BE* and record an error in the Error CSRs. As a bystander, Profile B modules shall ignore these transactions.

- Address Only Locked
- Copyback
- Invalidate
- Modified Response
- Read Invalid
- Read Locked
- Read Modified
- Read Partial Locked
- Read Shared
- Shared Response
- Write Invalid
- Write Locked
- Write No Acknowledge
- Write Partial Locked
- Write Response

Unsupported use of CA2*

Only a Read Partial or a Read Unlocked transaction may be split. If a master detects the assertion of CA2* (SR*) on any other transaction other than a Read Partial or Read Unlocked, the transaction ends with an error.

NOTE

Under Profile B, Read Response, Write Unlocked, and Write Partial transactions are not permitted to be split.

Continued on next page

Unsupported use of the ST[ ]* lines

The IEEE 896.1 protocol describes the Futurebus+ protocol for all profiles. Profile B does not support the following use of the ST[ ]* lines.

- The use of Wait (ST7* - WT*)
- The use of Intervention (ST5* - IV*)
- The use of Transaction Flag (ST4* - TF*)
- The use of Broadcast or Broadcall (ST3* - BC*)

Any Profile B module not participating in the transaction shall ignore the assertion of WT*, IV*, TF*, or BC*.

If a Profile B node sees any of the listed ST[ ]* signals on the bus for a transaction in which it is participating, the node should respond as indicated in Table 3–22.

Unsupported Use of the CM[ ]* Lines

In addition to the limited subset of Profile B commands available during the connection phase, certain uses of the command lines during the data and disconnection phases are also unsupported on Profile B.

- The use of Slave Write (CM4* during data, disconnect - SW*)
- The use of Lock commands (CM[2..0]* during data - LC[ ]*)
- The use of Lock (CM0* during disconnect - LK*)

Ordinarily, the situations that would lead to the unsupported use of these lines would also cause an error to be detected during the connection phase, either as an unimplemented command or an illegal status (such as IV*). If, however, the assertion of one of these signals occurred as a result of some form of transient error, the error should be reported as a *protocol error* and TE* and possibly BE* should be asserted.

Continued on next page

Table 3–22 ST[ ]* Bus Signal Responses

Phase	ST7* (WT*)	ST5* (IV*)	ST4* (TF*)	ST3* (BC*)
Master Response				
Connection	1	1	1	1
Data	2	3	3	2
Disconnect	2	2	2	2
Selected Slave Response				
Connection	2	2	2	2
Data	2	2	2	2
Disconnect	2	2	2	2

Key to Bus Signal Responses

1–Proceed to disconnection and set the appropriate error in the ERROR_HI CSR.
2–Ignore the assertion of this status line.
3–If in packet mode, check between packets; proceed to disconnection phase, and set the appropriate error in the ERROR_HI CSR; otherwise ignore.

Unsupported Cache Extension

The caching domain of the system bus does not extend to the Futurebus+ Profile B I/O bus. Caching functions are not required for correct operation of Profile B modules. Non-Profile B modules can establish a cache coherent domain on the I/O bus if they so desire. However, that caching domain does not extend to Profile B modules or across Profile B bus bridges.

Unsupported Message Passing

Message passing is not required for correct operation of Profile B modules. Non-Profile B modules in Profile B systems may use message passing but should not direct messages at Profile B modules.

Continued on next page

Unsupported Tag Bits

Profile B modules need not connect to the Tag bits TG[7..0]* and Tag Parity TP*. Parity checking and generation is not required and should not be done. The TG[7..0]* and TP* lines are bused and terminated on the backplane in Digital systems but are otherwise unimplemented in Digital systems and are RESERVED.

Unsupported Serial Bus

Digital backplanes bus and terminate the two Futurebus+ signals designated for Serial Bus. Serial bus is not implemented in Digital systems and is RESERVED.

Unsupported Address and Data Width

Digital systems support up to 64 bits of addressing on the Futurebus+. In accordance with the Futurebus+ specification, all modules must default to a 32-bit data path width upon initialization. The system configuration software then allows for a larger data path if the system support is available. All Digital systems bus and terminate at least AD[63..0]*.

Power Up

Types of Resets

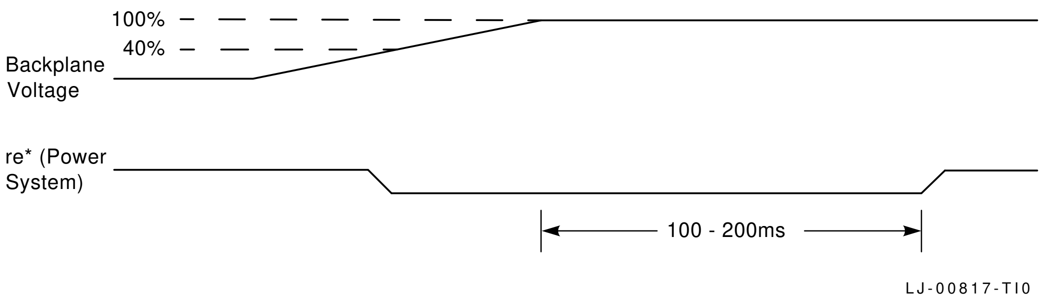
A Profile B module shall respond to the four different uses of RE*.

- Power Up
- System Reset
- Bus Initialization
- Live Insertion Alignment

Power Up

When a system is first powered up, the power system asserts the reset signal, RE* via the system bus bridge. The power system must assert RE* prior to any of the bus power voltages reaching 40% of their nominal value. The power system maintains RE* asserted for 100 to 200 ms after all of the bus power voltages are within regulation. This allows all modules on the bus to detect that a system reset is occurring. This power-up sequence is shown in Figure 3–12.

Figure 3–12 Power Up Sequence



System Reset

Definition

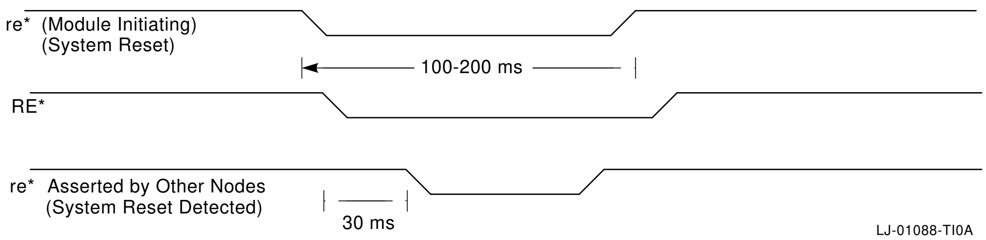
A System Reset can be directed at a single node by writing the RESET_START CSR. Refer to the RESET_START CSR section in the Architecture chapter of this handbook.

A System Reset can be directed at the entire system by asserting the RE* (reset) line for 100 to 200 ms. If RE* is not released within 500 to 1000 ms, all modules release RE* allowing the detection of a faulted reset line stuck asserted.

Sequence of Events

Figure 3–13 and Table 3–23 illustrate and detail the steps for a System Reset.

Figure 3–13 System Reset Sequence



Continued on next page

Table 3–23 Steps for System Reset

Step	Action	Description
1	Initiate	When the module initiates a system reset, RE* is asserted for 100 to 200 ms.
2	Monitor	All modules in a system monitor the RE* signal.
3	Assert	When a module detects that system reset is occurring, it asserts RE*.
4	Reset	The module then performs the internal reset operations necessary to allow participation in bus transactions.
5	Release	When a module completes these operations, it releases RE* indicating that it is ready to participate in transactions.
6	Perform	After releasing RE*, the modules perform Built-In-Self-Test (BIST). Note that modules may respond with Busy to CSR accesses while BIST is in progress.

Bus Initialization

Bus Initialization

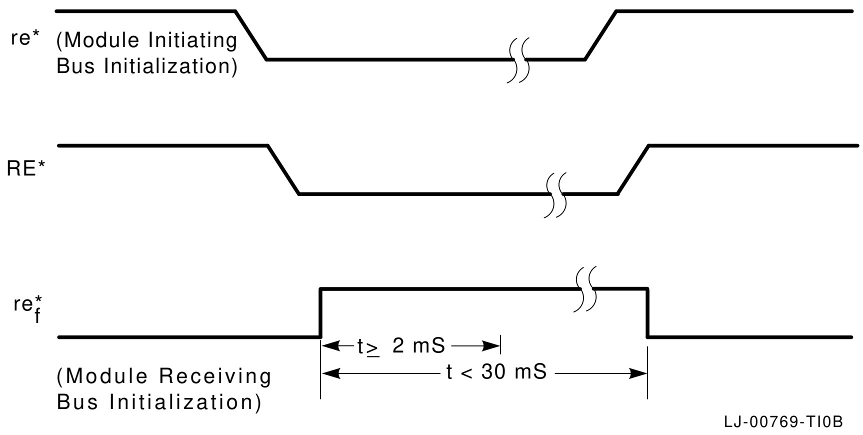
The following events take place during Bus Initialization:

- Modules detect RE* asserted for more than 2 ms.
- When a module detects a bus initialize operation occurring, it initializes its bus interface circuitry and waits to determine if it is also a system reset.
- If the module detects REf at logic zero prior to 30 ms, it resumes operation.
- Bus initialize resets the bus interface. It is transparent to the rest of the system. This may cause a system reset.

Modules shall not release AI* or AR* as part of the bus initialization process.

Figure 3–14 illustrates a bus initialization operation.

Figure 3–14 Bus Initialization Sequence



Live Insertion

Live Insertion Alignment

The Live Insertion feature of Futurebus+ is optional on Profile B modules. Digital systems *do not* support live insertion at the present time. However, Profile B compliance does require a module to recognize when another module is live inserted into a system.

Module Insertion

The logical protocols of Futurebus+ provide facilities for modules which are inserted into a powered operating system to align their logic state machines with the other modules on the bus and begin operation.

When a module is inserted into the bus and detects that it has just powered up, it checks to see if RE* is asserted. If it is not asserted, the module assumes it has been inserted into a live system. If the reset line is asserted, it waits to see if it is a full system reset and not a bus initialize or live insertion. Eventually, the module detects if it is being inserted into a live system.

A module inserted into a live system asserts RE* to notify all the other modules that it needs to align. All other modules upon detecting RE* asserted should limit the remaining length of the current parallel bus transaction to 128 μs and wait. This causes the bus to go into an idle state. The inserted module waits after it has asserted reset until it detects the bus in a continuous idle state for at least 1 μs and then it asserts AI* and AR* to complete the alignment. It then releases RE* and all modules resume operation when they detect REf at logic zero.

Chapter 4
Physical Layer

Physical Layer Overview

Overview

The Physical Layer chapter describes the electrical and mechanical aspects of Futurebus+ Profile B. Profile B is an explicit physical implementation of the IEEE 896.1 standard.

Topics Covered

This chapter covers the following topics:

- Futurebus+ topology
- Transceivers, signal characteristics, and noise margin
- I/O module (includes the front panel and PWB)
- Connectors
- Card cage
- Power supply
- Electrostatic discharge (ESD) concerns
- Electromagnetic compatibility (EMC) concerns

These aspects claim conformance to Futurebus+ Profile B as defined in the IEEE standards, 896.2, 1301, 1301.1, and EIA IS-64.

Metric Dimensions

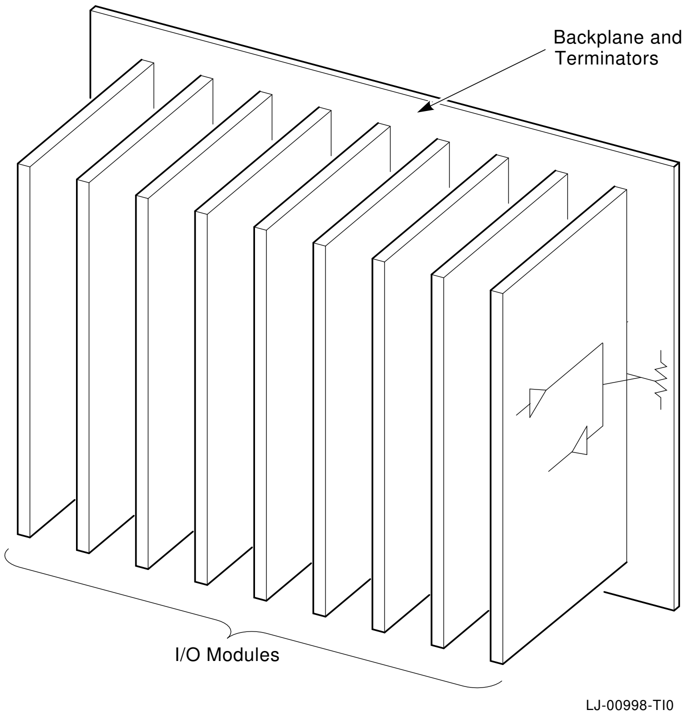
Profile B is a *hard metric* standard as defined by IEC 916 and IEC 917.

Futurebus+ Topology

Physical Topology

The bus is physically arranged as a conventional bidirectional multidrop configuration with up to 14 slots. Figure 4–1 illustrates the physical configuration of the bus.

Figure 4–1 Bus Structure

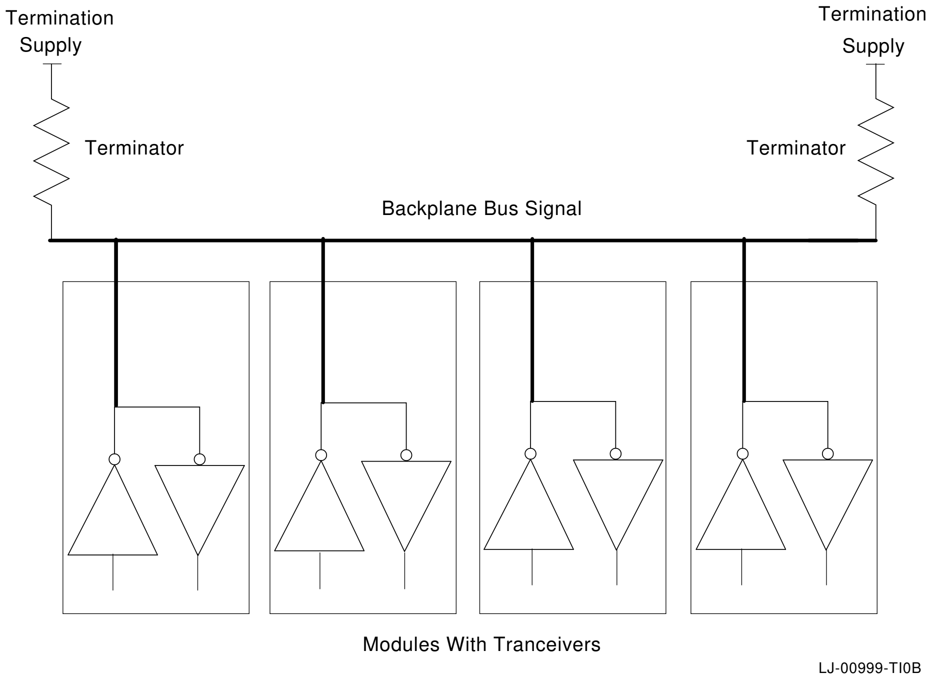


Continued on next page

Electrical Topology

Figure 4–2 illustrates the electrical view of the bus.

Figure 4–2 Electrical View of Futurebus+ Profile B

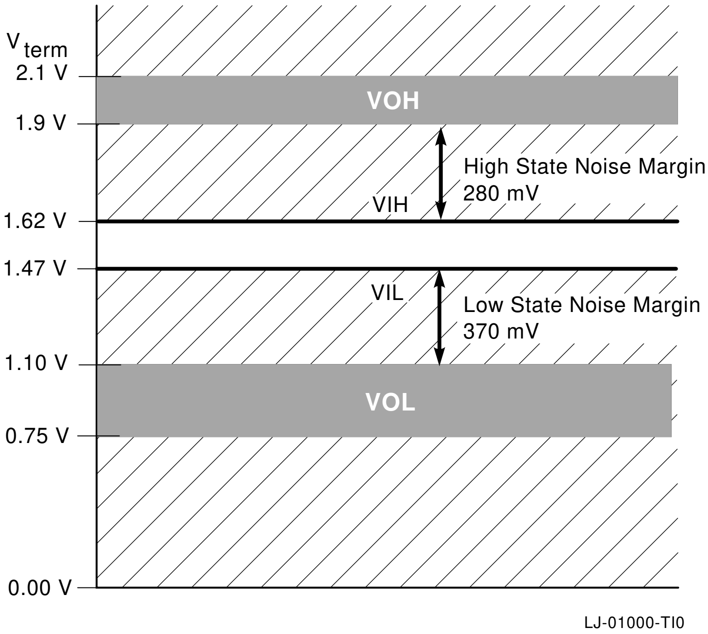


Futurebus+ Signal Characteristics

BTL Transceivers

All electrical signaling on the bus must use Backplane Transceiver Logic (BTL) voltage levels. The two BTL voltage levels are approximately 1 V and 2 V (for an approximate 1 V logic swing). BTL also incorporates negative logic on the bus (see Figure 4–3). For further information, refer to the IEEE 1194 standard.

Figure 4–3 BTL Voltage Levels



Bus Signal Requirements

All Futurebus+ signals require controlled impedance. Each bus signal has a termination resistance at each physical end of the bus (see the Backplane section of this chapter for reference).

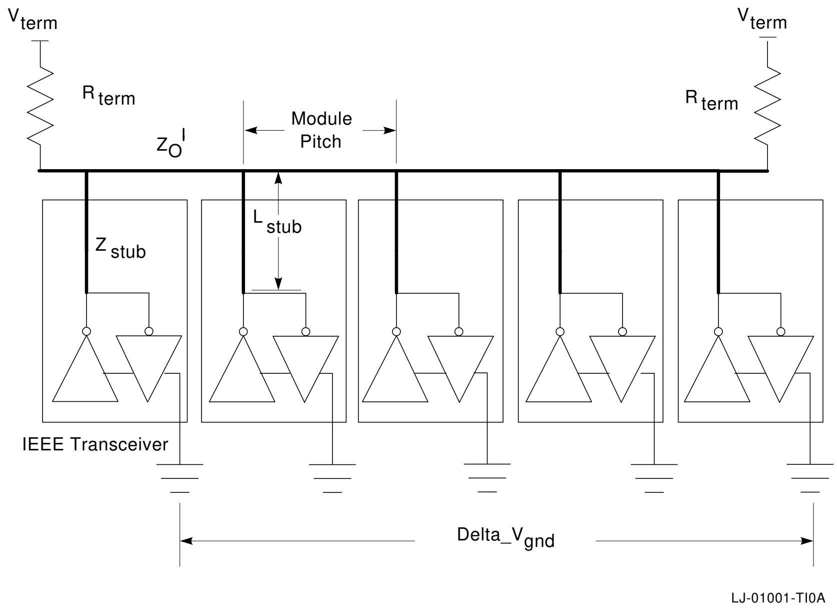
Modules introduce stubs onto the bus. These stubs must be carefully controlled (see Figure 4–4).

Continued on next page

Electrical Parameters

This section defines the Futurebus+ electrical parameters. Figure 4–4 illustrates how some electrical parameters are defined.

Figure 4–4 Electrical Specification Parameters



Continued on next page

Definitions

Z_0: The characteristic impedance of backplane bus signals without connectors installed. This assumes connector plated through holes (PTH) are present.

$Z_0{'}$: The characteristic impedance of backplane bus signals without connectors or termination resistances installed. This assumes only signal and return PTH, and all conducting planes are present with the corresponding dielectric insulating layers.

Rterm: The resistance of the bus terminator. A terminator is located at each physical end of the bus signal, most likely mounted directly on the backplane.

Rsig: The maximum total resistance of a bus signal PTH on the backplane.

Vterm: The bus termination voltage. This is the supply voltage for the Rterm terminators. This value is critical due to the sensitivity of high-state noise margins to this value.

delta_Vgnd: Maximum backplane reference (ground) plane difference measured across the bus backplane.

Signal_skew: All bus address/data signals must propagate along the bus at the same speed (or slower) than related strobe signals. This ensures strobe signals will latch data/address signals. Signal_skew defines a maximum of how much slower strobe signals can be as compared to address/data signals.

Zstub: The characteristic impedance of the signal line stub on a module.

Lstub: Maximum length of a module bus stub. Lstub is the stub length that is measured from the connector PTH to the center of the transceiver pin surface-mount pad (include etch length from any dispersion PTH).

Tr, Tf: Minimum (fastest) transition time (rise and fall times) allowed by any logic connected to the bus.

Continued on next page

Module pitch: Distance between module slots along the backplane.

Number of slots: Maximum number of backplane slots where electrical operation is guaranteed.

Summary of Electrical Specifications

Table 4–1 provides a summary of the electrical specifications.

Table 4–1 Summary of Electrical Specifications

Parameter	Min	Nominal	Max	Units	Comment
Z_0	55	60	66	Ω	Bare backplane including PTH.
Z_0'	52	57	62	Ω	Connectors installed, no terminators or modules installed.
Rterm	-	33	-	Ω	Termination resistance. (Rterm is ±1%.)
Rsig	-	-	1.0	Ω	Series resistance.
Vterm	-	2.1	-	V	Local bulk and filter capacitance may be required. See IEEE 896.2, Chapter 7, Voltage Variations section. (Vterm is ±2%.)
delta_Vgnd	-	-	50	mV	Limits ground-plane voltage drop under maximum current load.
Skew	-	-	675	ps	Strobe signals should never be faster than corresponding address/data signals.
Module Pitch	-	30	-	mm	-
Zstub	45	58	75	Ω	-
Lstub	-	-	25.4	mm	-

Refer to IEEE 896.2 for the most up-to-date information on electrical parameters.

Futurebus+ Noise Margins

Overview

To ensure positive noise margins for the Futurebus+ bus signals, specific electrical requirements must be met by module and backplane designers for the following components:

- Backplane
- Connector
- Module
- BTL Transceivers

Module Conditions

The module requirements for positive noise margins are:

- Stub length = 25.4 mm (1") or less.
- No more than two PTHs per stub; one of the PTHs may be at the transceiver and the other at the connector.
- One transceiver per stub.
- Stub Z_0 = 45 to 75 Ω.

Backplane Conditions

The backplane requirements for positive noise margins are:

- 14 slots (maximum)
- 30 mm module pitch (module-to-module spacing), 150 mm (maximum) module pitch (5 empty slots, maximum)
- +2.1 V supply must be within the range of:
  +2.1 V ± 2% and no more than ± 50 mV of ac ripple, noise, and so on
- Uniform distribution of plug-in modules in partially populated systems (no more than 5 contiguous empty slots between modules)

Futurebus+ Multiple Data Widths

Module

Profile B allows designers to build modules that connect to 32, 64, or 128 data-bit wires (A/D transceiver bits). All modules must support 32-bit width; 64 and 128 bits are optional. Modules that do not support 128-bit data or I/O PTH connector group "E" can omit connector group E. Similarly, backplanes must support 64-bit data widths and optionally support 128-bit data width.

For Futurebus+ modules with an address or data width wider than the actual width of the backplane into which the module is inserted, transceivers that do not connect to bus lines must be set to drive, not receive. At powerup, modules *must* assume a 32-bit bus width. (This is set using the LOGICAL_MODULE_CONTROL CSR.)

Futurebus+ Multiple Address Widths

Modules and Backplanes

All Futurebus+ modules and backplanes support 32-bit (A32) addressing. In addition, modules and options can optionally support 64-bit (A64) addressing.

Futurebus+ Profile B Implications

PWB Physical Dimensions

These modules adhere to the IEEE 1301 standard as 12 SU x 12 SU x 6 SU modules.

Profile B is a hard metric standard, as defined by IEC 916 and IEC 917. The smallest indivisible grid on the PWB (and backplane) is defined (0.5 mm). All features on the boards are specified in relation to this grid.

At the next higher assembly, the module, adapter, or plug-in unit, the smallest indivisible grid is defined (5.0 mm). All features on the boards are specified in relation to this grid.

At the next higher assembly level, the card cage, the smallest indivisible grid is defined:

5 mm in the horizontal (X) direction
25 mm in the vertical (Y) direction
25 mm in the depth (Z) direction

All dimensions are specified in relation to this grid.

In addition, the ratios of the indivisible dimensions across assembly units are decimal numbers 1, 2, 5, 10, and so forth. This satisfies the *modular order* requirement of IEC 916.

Number of Modules

Profile B supports up to 14 modules. Not all Profile B systems provide this many slots. Not all systems provide sufficient power for all possible combinations of modules.

Futurebus+ Arbitration

Central Arbitration Signals

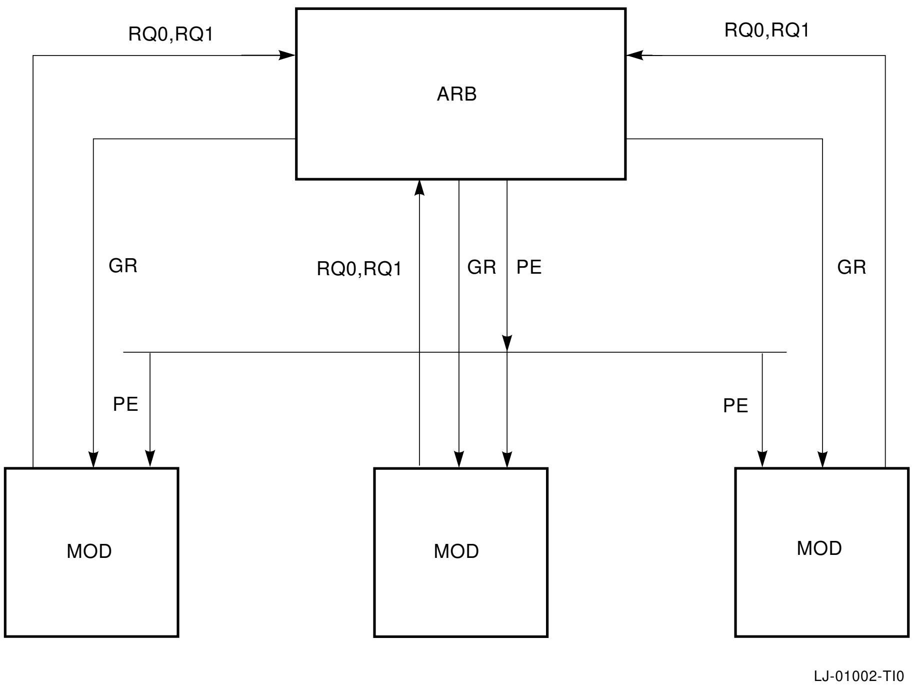
All Digital Futurebus+ systems built for Profile B compatibility use central arbitration (see Figure 4–5). The central arbitration logic could reside in various locations. A few options are as follows:

- On the backplane as active logic
- On a backplane *daughter card*
- Sharing a module slot with the bus bridge

Current Digital implementations have chosen a merged bridge /arbitration module scheme.

The arbitration signaling scheme is illustrated in Figure 4–5.

Figure 4–5 Arbitration Block Diagram



Continued on next page

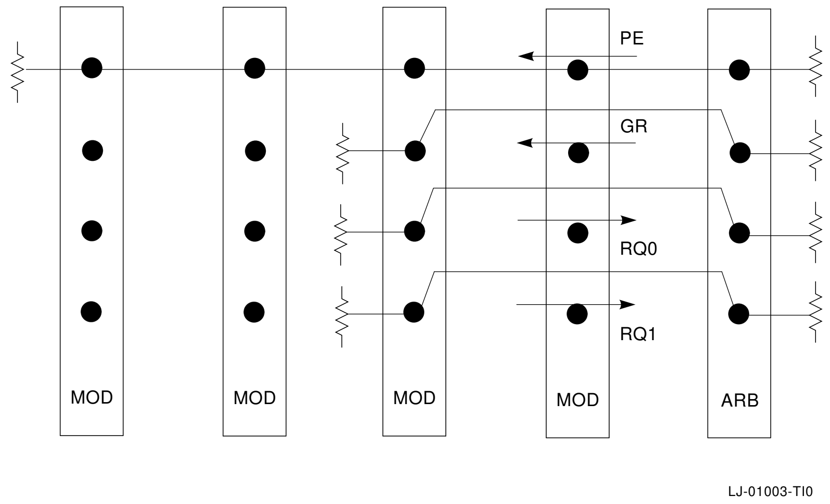
Arbitration Signaling

The IEEE 896.1 standard defines a set of central arbitration signal lines: Request (RQ[1..0]*), Grant (GR*), Preempt (PE*), and End of Tenure (ET*). These signals are BTL (IEEE 1194.1 signal levels). Signals RQ[1..0]* and GR* are radially connected from the central arbiter to each Futurebus+ slot on the Profile B backplane. The PE* signal is bused to all Futurebus+ slots on the Profile B backplane.

Electrical Operation

The electrical operation of arbitration signals is the same as all other bus signals. That is, arbitration signals are driven with BTL devices and are terminated on the backplane. Note that point-to-point arbitration signals are terminated at both ends. Termination resistors for point-to-point signals are not required to be mounted at the physical extremes (edges) of the backplane. These terminations can be placed at the physical end of the signal at each module instead (see the RQ0*, RQ1*, and GR* signals in Figure 4–6).

Figure 4–6 Arbitration Signaling



Module Transceivers

Description of Transceivers

The BTL transceivers drive and receive information to and from the backplane bus. The transceivers conform to the IEEE P1194.1 (Backplane Transceiver Logic) standard and the IEEE P1156 (Environmental) standard.

The BTL transceivers perform a wire-OR function on all bus signal lines. They are active-low on the Futurebus+ side. See Figure 4–2, Electrical View of Futurebus+ Profile B, in the Signal Characteristics section of this chapter.

Output Drive

The quad flat pack (QFP) package of the transceivers is pinned to allow all bus signals to escape from one side of the package. This allows placement to minimize stub lengths. For Profile B systems, the maximum allowable stub lengths are 25.4 mm (1"). For QFP packages, this requires a double-sided module placement of the transceivers.

Mounting

Transceivers are mounted using surface mount technology. Current pin densities for QFP devices have pin-to-pin spacing of approximately 0.635 mm (25 mil). Package-to-package spacing can be as close as 0.635 mm (25 mil) measured from tip-to-tip of adjacent pins.

Signal Characteristics

In accordance with the IEEE 896.2 and 1194.1 standards, the BTL transceivers must have logic level and slew rate characteristics compatible with Futurebus+ requirements. These are described earlier in the Futurebus+ Signal Characteristics section of this chapter.

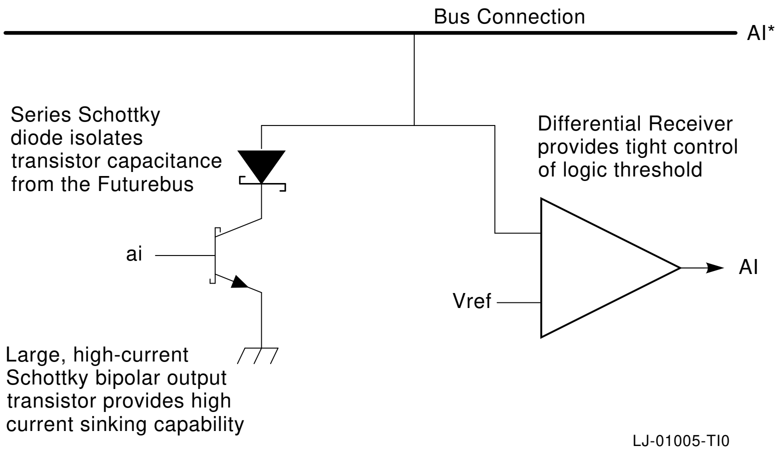
Continued on next page

BTL Summary

Figure 4–7 illustrates a typical BTL transceiver. The characteristics of BTL are as follows:

- Separate BTL and TTL ground connections
- Signal swing of $\approx$1 V
- Differential amplifier for receiver
- Precision narrow-threshold receiver input
- High-current bus drivers with 80 mA to 100 mA current sink capability
- Isolation of driver transistor capacitance from bus with Shottky diode
- Power dissipation makes it difficult to integrate more than 9 drivers in one package.

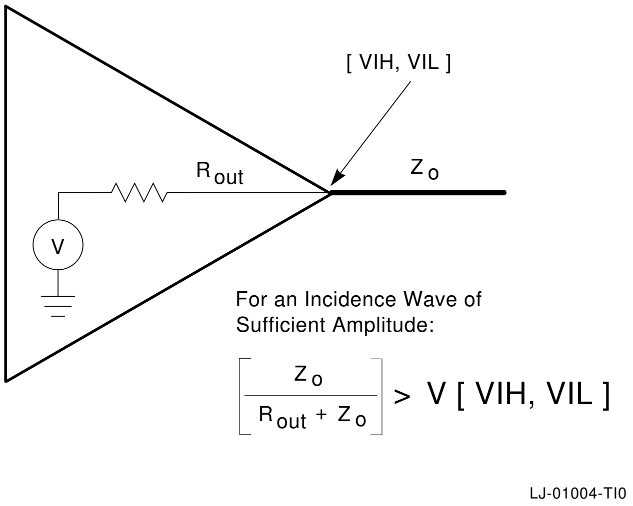
Figure 4–7 Logic Levels



Continued on next page

For properly designed systems, BTL offers *incident-wave* switching (see Figure 4–8).

Figure 4–8 Incident Wave Switching



Capacitance

A Futurebus+ transceiver presents less than 5 pF of capacitive loading at the QFP package pin because of the Shottky isolation diode.

Cross-Talk Avoidance

Cross-talk between Futurebus+ signals can be minimized by taking the following steps:

- Provide good grounding to minimize *ground bounce*.
- Minimize distance that multiple conductors travel in close parallel proximity.

Continued on next page

Module Transceivers, Continued

Stub Length

Zstub is the characteristic impedance of the signal line stub on the module. Lstub is the stub length that is measured from the connector PTH to the center of the transceiver pin surface-mount pad (include etch length from the dispersion PTH, if present).

The maximum stub length is equal to or less than 25.4 mm (1 inch).

Placement

To meet electrical stub length requirements, transceivers are placed as close as possible to the signal connectors. Also double-sided mounting is necessary with QFP packages.

Glitch Filter

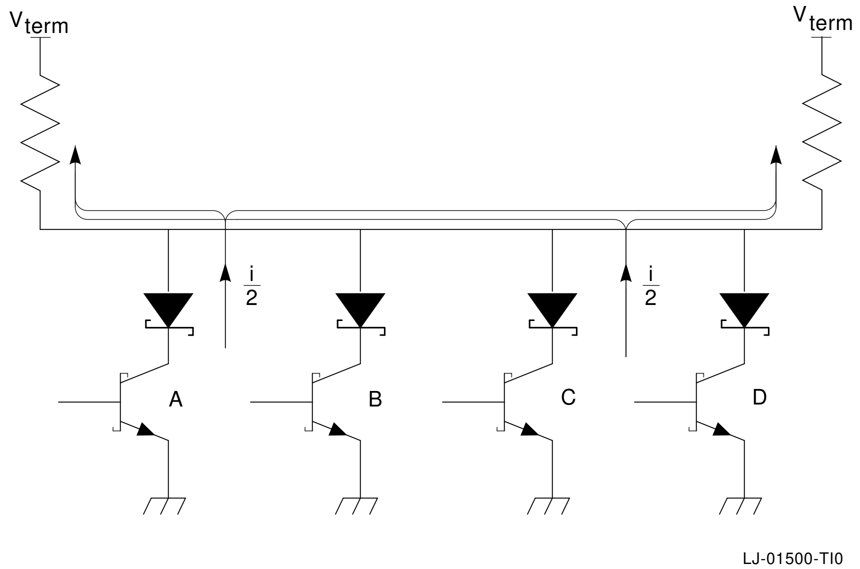
All buses that use drivers connected in a wired-OR configuration (such as the Futurebus+) are subject to a physical phenomenon known as *wire-OR glitch*. This glitch occurs when multiple drivers have been pulling a line down and one of the drivers subsequently switches *off*.

Development of the Glitch

Assume that drivers A and C are both asserted, therefore, they are both pulling the bus line low. The current from the terminators is being shared between the two asserted drivers (see Figure 4–9).

Continued on next page

Figure 4–9 Initial Steady-State Condition (Two Drivers On)

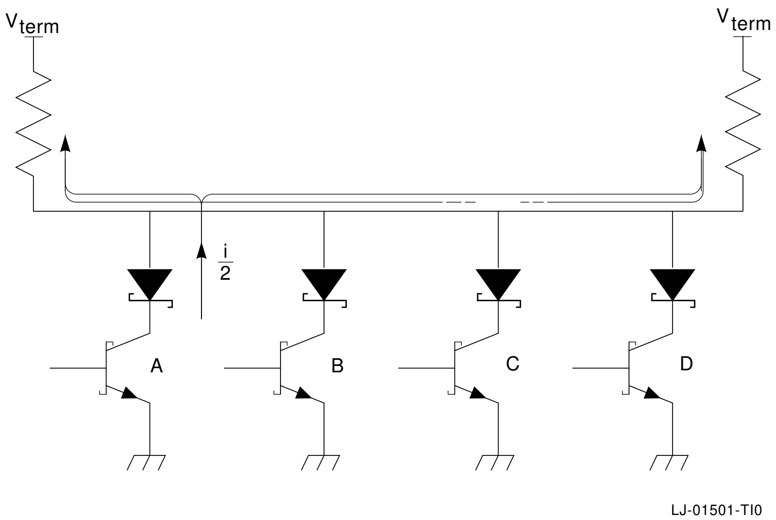


Continued on next page

Development of the Glitch (continued)

Assume that driver C releases. Now the entire current from the terminators must flow into driver A (see Figure 4–10). Because the bus is a transmission line environment, this effect does not happen instantaneously. Instead, the voltage along the bus wire will rise momentarily while the current flow shifts entirely into driver A. Eventually, driver A is supplying all of the current and steady-state conditions return (see Figure 4–11). It is this momentary rise in the voltage during the time that the current flow is changing that represents the wire-OR glitch (see Figure 4–12).

Figure 4–10 Transient Condition (One Driver On)



Continued on next page

Figure 4–11 New Steady-State Condition (One Driver On)

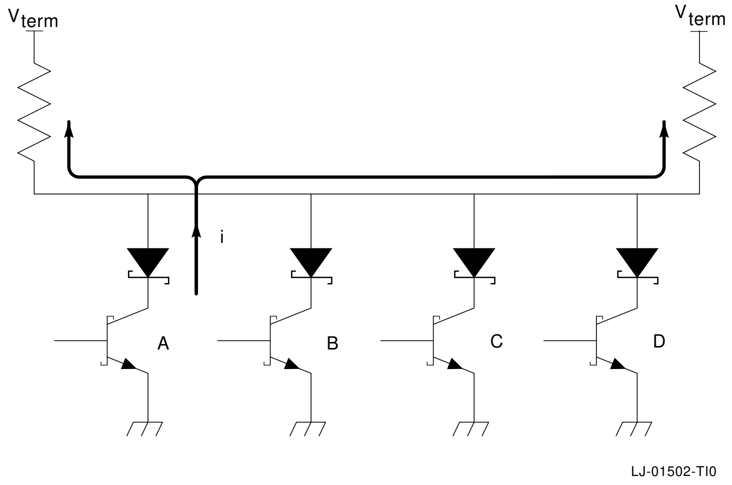
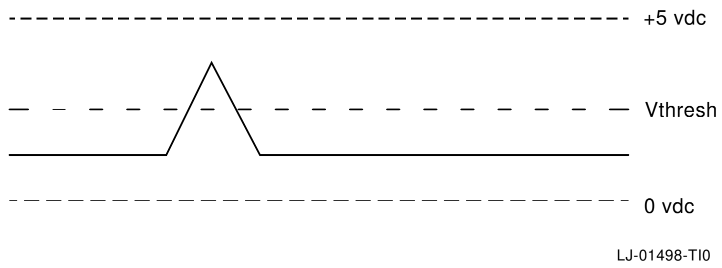


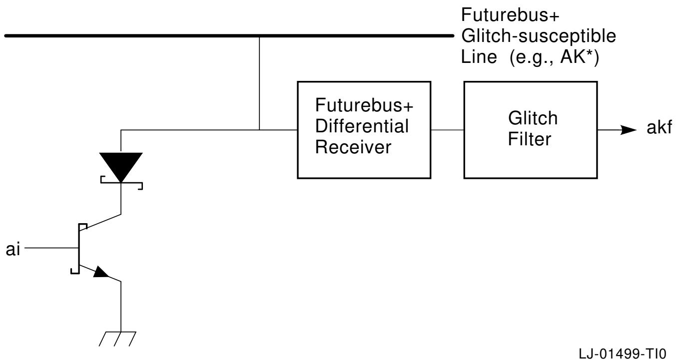
Figure 4–12 Wire-Or Glitch (Voltage Versus Time)



To avoid inducing errors, the output of the receivers of certain Futurebus+ signals must be filtered to eliminate these glitches. The glitch filters reject the short-duration, positive-going pulses caused by the wire-OR glitch (see Figure 4–13).

Continued on next page

Figure 4–13 Futurebus+ Glitch Filter



Glitch filters are required for AK*, AI*, and RE*. Glitch filters are required for AP*, AQ*, and AR* if the module receives these signals. The IEEE 896.2 standard mandates the use of glitch filters in systems that are implemented using the signaling environment of BTL.

Module Power Consumption

Power Limitations

Module power consumption is limited in two different ways:

- Current limit
- Cooling limits

Current Limits

In addition to the overall limits on the amperage available from any given power supply rail, a limit is imposed by the current rating of the connector blades that supply power to each I/O module. Therefore, the number of power blades at the connector and their current-carrying capacity define an absolute upper bound on current and power for each module. Individual blades are rated at 2.75 A each; Digital recommends derating to 2 A (max) current for each power blade to allow for uneven current sharing among the blades. This limits the available power at each voltage to the following:

5	V source:	8 blades,	16 A,	80 W
3.3	V source:	8 blades,	16 A,	53 W
48	V source:	2 blades,	4 A,	192 W

Cooling Limits

The total power per slot shown exceeds the cooling provided by the enclosure, and in general, exceeds what the system will have available for each slot (see the ''Cooling and Power Dissipation'' section of this chapter).

Input Voltage Choices

For the following two primary reasons, 48 V power is provided:

- To allow the development of voltages not provided from the backplane
- To support live insertion of modules by using switched regulators powered from 48 V.

Continued on next page

Module Power Consumption, Continued

Input Voltage Choices (continued)

This allows the developer some latitude in the choice of power (voltage) source. Keep in mind that the available power at each voltage is both system-dependent and reduced by the other loads in the system.

Power Rails

Backplane power pin assignments provide for the following six power rails:

+5.0 V
+3.3 V
+V (48 V DC supply)
-V (48 V DC return)
VBP
0 V DC

The +5.0 V, Volt Backup Power (VBP), and +3.3 V supplies are regulated voltages and use 0 V DC for power return. The system power supply may tie either the positive or negative 48 V terminal, or any reference in between, to 0 V DC. Module designers developing products that use 48 V, therefore, must provide isolated inputs because the 48 V reference will differ between system enclosures.

Power Sequencing

The power sequence supplied by the power supply is described later in this chapter. If the module requires another sequence, the module must contain on-board provisions such as Field Effect Transistors (FETs) to generate the required sequence.

Bypass Capacitance Limits

The maximum capacitance a module may present to the 5.0 V supply is 1000 μF, to the 3.3 V supply is 1000 μF, and to the VBP supply is 250 μF.

Continued on next page

Power Connections

For any voltage being used by a particular module, connection shall be made to all power blades assigned to the voltage. See the IEEE 896.2 standard, Profile B Connector, Power, and Signal Pin Assignment section. This connection shall be made through a low impedance to assist uniform current sharing through all blades.

If a supply voltage is not provided by a particular system power supply, the pins assigned to that voltage shall be reserved. However, if battery backup is not provided, the VBP shall be connected in the backplane to +5.0 V if +5.0 V is present in the system. Modules shall not connect VBP to the +5.0 V or +3.3 V rails on the module. In addition, 0 V DC power returns and 0 V signal grounds shall be connected to backplane ground plane(s) at all connector positions in the backplane.

Connector Power Blade Current Rating

The current passing through any power supply blade or power return blade shall not exceed 2 A (averaged across all blades) for each supply voltage.

Module Current Specification

For Profile B modules, published specifications for module current demand at each supply voltage are based on a calculation using both the typical (I_{typ}) and maximum currents (I_{spec}) specified for each active component on the module:

$$I_{max} = [I_{typ}{}^2+(I_{spec} - I_{typ})^2]^{0.5}$$

This formula avoids grossly overstating the power dissipation of a module built from a mix of typical components.

The maximum current of a Profile B module at a given supply voltage is the sum of the I_{max} values for active devices plus nominal currents for passive components. Maximum power is the sum of the products of maximum current times the corresponding nominal voltage.

Continued on next page

Module Power Consumption, Continued

Module Current Specification (continued)

Maximum current for each voltage and maximum power shall be declared in module product literature and product specification for Profile B modules. See the IEEE 896.2 standard, Profile B Power section, for further details.

48 V Requirements

On modules, power shall be drawn from the +V and -V only through galvanically isolated DC-DC converters. Modules using power from +V and -V shall be able to operate with +V and -V within ranges recommended by the IEEE 896.2 standard, Voltage Variation section. On any one module, the maximum DC leakage current from the 48 V supply is 5 μA. This current is defined as current sourced from the +V and not returned to the -V or vice versa. Proper design practices should be taken to ensure that digital or analog signals derived from the +V/-V supply do not capacitively couple to the system logic or backplane ground and that such signals be separately shielded with a separate return path to the single point or primary system ground.

SELV Requirements

All voltages in operator accessible areas shall meet the SELV isolation (creepage, clearance, insulation strength, distance through insulation) requirements of the safety standard recommendations specified in IEEE 896.2.

Modules using 48 V must be designed carefully if they are to conform to Underwriters Laboratories (UL) LPWR limitations and other relevant standards (UL 1950/IEC 950 SELV, etc.). This is because the presence of 48 V on the module makes it easy to exceed the 60 V peak-to-peak maximum voltage requirements of SELV designs.

Module Decoupling Requirements

Decoupling Capacitors

Decoupling and storage capacitance is necessary to maintain power supply voltage tolerances during all logic switching.

Two high-frequency bypass capacitors are required for each transceiver. Transceiver decoupling capacitors should be placed as close as possible to the transceivers. A bulk storage capacitor is also required to accommodate transceiver switching currents.

NOTE

Users are required to provide the necessary bulk and high-frequency decoupling capacitors for their own logic as well as for the transceivers.

Bypass capacitors connected to GA lines must conform to the characteristics as required in IEEE 896.2, Chapter 7, Geographical Address Electrical Characteristics.

Care should be taken to add sufficient bypass and storage capacitance to accommodate the additional switching current loads. Calculating bypass and storage capacitance requirements for a module is a complicated task. It is best to overestimate capacitance requirements if actual switching loads are not known.

The maximum capacitance a module may present to the 5.0 V supply is 1000 μF, and to the VBP supply is 250 μF.

Recommended BTL Transceiver Grounding

The BTL transceiver packages contain multiple ground connections for the Futurebus+ pins. These multiple ground connections nearly eliminate ground bounce and similar effects from simultaneously switching outputs. Additional grounds are also allocated to the TTL side.

Although there are many ground connections, it is recommended that adequate decoupling capacitors be allocated for each transceiver.

Cooling and Power Dissipation

Environment

Profile B systems target computer room, offices, and remote (nonattended) environments. The computer room environment is the most benign, offering the best temperature and humidity control, as well as high tolerance for acoustic noise. The office environment places requirements that are incompatible with the remote applications. This environment offers temperature control that reduces heat and cooling problems but a low tolerance to air mover noise. The nonattended environment, which has higher temperature extremes, allows higher noise levels. Both environments are typically satisfied with the same system through the use of fan speed control, where the air mover angular velocity tracks inlet air temperature between two set limits.

Airflow

As a minimum, Profile B systems provide the following levels of cooling for all values of back pressure below 4 Pa.

- At 45°C, modules are provided with a minimum airflow of 1.5 linear m/s (300 linear ft/min).
- At 25°C, modules are provided with a minimum airflow of 1.0 linear m/s (200 linear ft/min).
- Between these two levels, modules are provided with airflow that is at least the straight-line derated value appropriate for that temperature.

  The airflow sets an upper bound on module power dissipation and corresponds to a potential dissipation of 80 W/slot for a bulk air temperature rise of 10°C. However, it is unlikely that system designers will provide sufficient power to support a simultaneous 80 W load at every slot in the system. (The rating of the power supply represents a fixed cost that does not scale with use, therefore, the system developers will only provide what they believe their systems will use.)

Continued on next page

Cooling and Power Dissipation, Continued

Airflow (continued)

If additional airflow is required, the module specifications must explicitly cite this need.

The uniformity of the air velocity between the front panel and the backplane connector shall be $\pm 10\%$ in an empty slot (the components mounted on a plug-in unit will disturb this distribution).

Thermal Design

The limit on power dissipation is presumed to be the junction temperatures of the hottest components and their effect on MTBF. These temperatures should be modeled and measured. The corresponding MTBF and its method of calculation can then be part of the specifications of the option.

In general, good thermal designs should be able to achieve 80 W module dissipation without the use of heat sinks, unless the power dissipation is concentrated in just a few hot spots.

To eliminate thermal coupling between modules, it is sufficient with air velocities above 1.0 m/s to limit power dissipation on side 2 of every module to less than 20 W.

Environmental Conditions

The target applications previously described led to the selection of IEEE 1156.1, Class 4, as the appropriate environment for both Profile B systems and plug-in units. The plug-in unit test limits are shown in the following tables.

Non-Operating Conditions

Table 4–2 Non-Operating Conditions

Test Parameter	IEC Test Publicat	Severity /Conditions	Requirements
Low temp	63-2-1-AB	24 h	-40°C
High temp	68-2-2-Bb	24 h	+65°C
Thermal shock	68-2-14	<3 min	-55° to +70°C
Humidity	68-2-30-Dd	6 cycles	+25°C to +55°C 93% RH
Shock	68-2-27-Ea	6 ms	30 G
Flame A	695=5=5	Extinguish	<30 s
Flame B	ASTM-D2863		OI>=28% O_2
ESD	801-2		30 kV/150 pF

Continued on next page

Operating Conditions

Table 4–3 Non-Operating Conditions

Test Parameter	IEC Test Publications	Severity /Conditions	Requirements
Low temp	63-2-1-Ad	24 h	+5°C
High temp	68-2-2-Bb	24 h	0.35 mm/5 g
Sine vib C	68-2-6-Fc	5 - 100 Hz	0.35 mm/5 g
Shock	68-2-27	18 pulses	15 g/11 ms
Low air press	68-2-13	16 h	1050/700 mbar
ESD A	801-2		15 kV/150 pF
EMI	FCC, 15, J		MEET

Printed Wiring Board (PWB)

Description

The PWB provides the mechanical support for the IEEE EIA IS-64 connector, the front panel, the Futurebus+ transceiver, any necessary additional logic or functionality, and for the hardware that protrudes through the front panel. In addition, the PWB rides on the card guides to both guide and support the entire plug-in unit. It contains an ESD strip, and serves to channel air flow. To implement the Futurebus+ interface, a minimum of 6 layers is required (1.6 mm thick, nominal); however, the specifications allow up to 10 layers (2.4 mm thick, nominal).

Dimensions

The location of holes for mounting the front panel may differ depending on the vendor of the front panel and the size and location of the I/O connectors. The IEEE 1301 standard defines the standard location of these mounting holes, which are located to satisfy the 25 mm grid restrictions of the subrack.

Near the EIA IS-64 defined connector pin-hole field are three 6.0 mm holes with clearances defined. These holes and clearances allow the addition of one of the two keying schemes previously described and defined in the standard.

Special care should be taken in the hole-drilling operations to ensure that there is no build-up of round-off error during the step-and-repeat operations as a result of converting the 2 mm spacing between holes from metric to its closest imperial equivalent.

Connector Mounting Detail

Figure 4–14 and Figure 4–15 show the field of holes for mounting the EIA IS-64 connector to the PWB, including tolerances.

Figure 4–14 PWB Detailed Dimensions

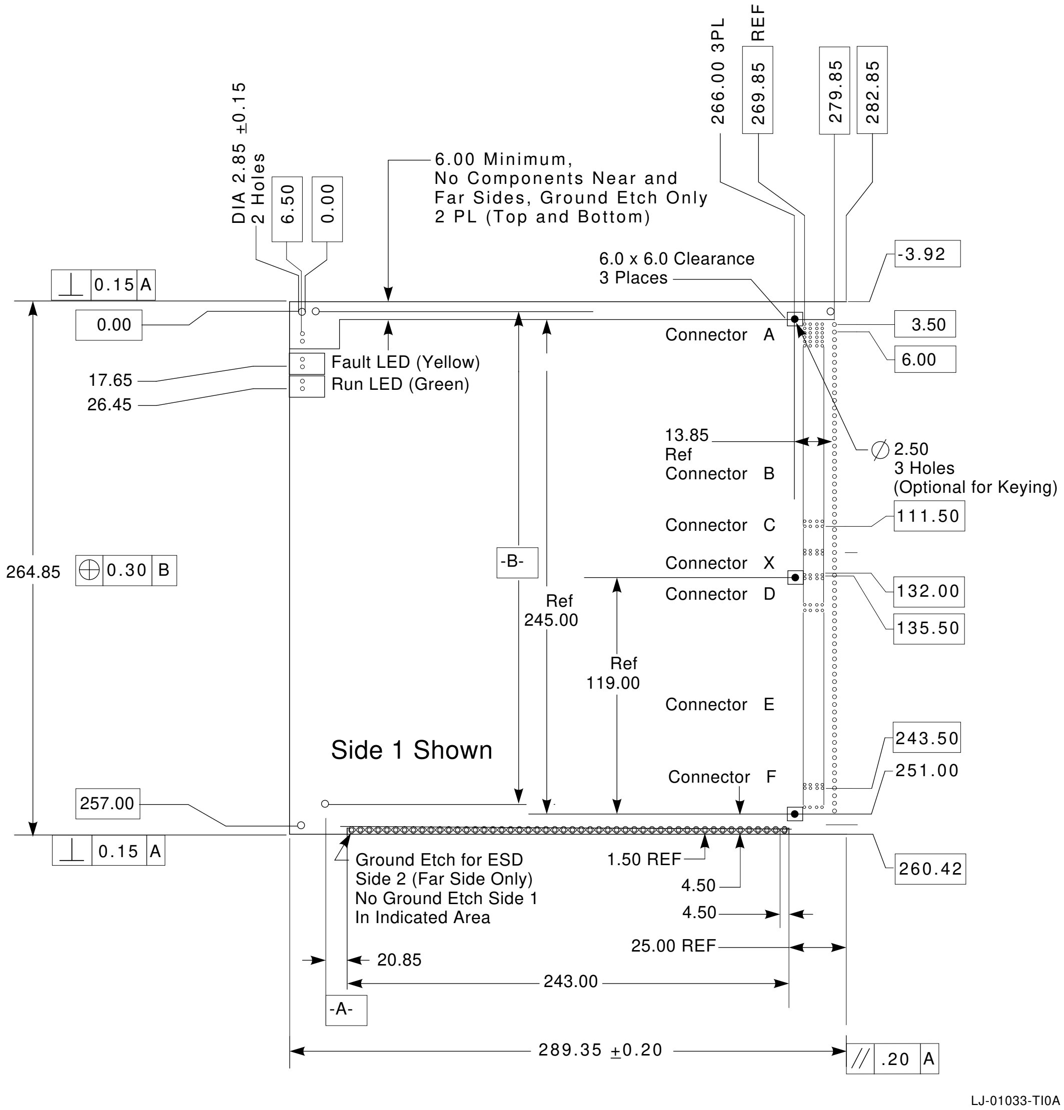
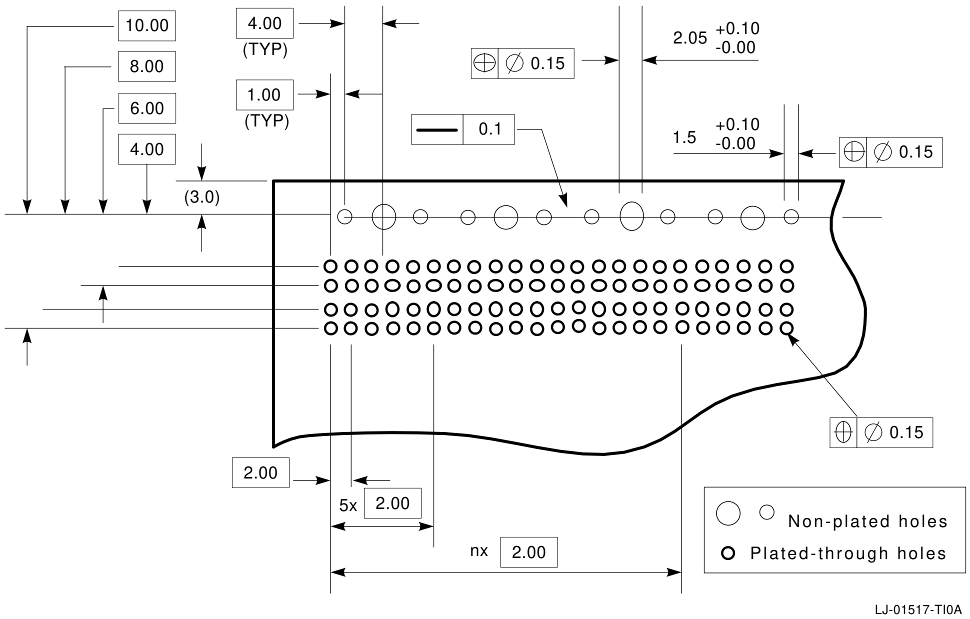


Figure 4–15 Recommended PWB Mounting Pattern Board Connectors



Module PWB Requirements (Module Layups)

Signal Impedance

Stub signal impedance must be in the range of 45 to 75 Ω.

Z_0 Characteristic Impedance

The PWB can be designed to allow controlled impedance characteristics. The transceiver stubs have a broad impedance range of 45 to 75 Ω. This parameter is not critical, however, higher stub impedances do provide slightly better noise margins. Etch impedance for general purpose logic should be optimized for the particular logic family used. TTL and CMOS signals generally require module signal etch impedance of at least 68 Ω.

Module Layout (Component Placement)

There are three principle areas on a Futurebus+ module: the transceiver area, the user area, and the I/O area. There is also a small *keep out* area along the edges of the module (see Figure 4–14).

Transceiver Area

The transceiver area contains Futurebus+ signal connectors, power connectors, transceivers, decoupling capacitors, and miscellaneous resistors. The connectors must be precisely placed on the module to comply with IEEE 896.2 mechanical specification positions.

User Area

Users can place components anywhere in the region outside of the transceiver area. Care should be taken not to exceed the component height restrictions. Component heights differ on each side of the module (see Figure 4–18 in the Module Mechanical Specifications section of this chapter).

Component package pin density determines the number of routing layers required for the module. Sufficient power supply decoupling capacitance for user logic must be provided.

Continued on next page

I/O Area

All I/O connections must be made through the bulkhead/front panel area. Users may position I/O connectors anywhere along the front panel within the user-defined area as specified in the IEEE 896.2 standard. The front panel mounting hardware may require customization (see Figure 4–16 in the Module Mechanical Specifications section of this chapter).

Keep Out Area for Card Guides

The component keep-out area is 6 mm wide to prevent interference between the module components and the card guides (see Figure 4–14). The only etch allowed within the card guide reserved area is one ESD pad on side 2 at the location indicated in this figure.

Module ESD Strip and Resistor

ESD Pad

The module electrostatic discharge pad on the PWB is designed to make contact with the discharge clip located in the card cage.

The module electrostatic discharge pad contacts the card cage discharge clip immediately after the module enters the card cage, and maintains contact thereafter during insertion, operation, and until the module is withdrawn.

ESD Implementation Rules

A 1 MΩ resistor is placed between the module ESD pad and module power return (0 V DC) to limit any static discharge current.

Static discharges that occur directly to the front panel are required to use the card cage as the discharge return path. For this reason, the front panel is made of conductive material. The PWB is not intended to carry static discharges from the front panel.

Module Mechanical Specifications

Description

The Profile B module is designed to fit in a 12 SU x 12 SU card cage, as defined by IEEE 1301. The module uses the 2 mm pitch metric connector described in IEEE 1301.1 and specified by EIA IS-64. However, the connector field, both in pin lengths and distribution of power and signal pins, is unique to IEEE 896.2 Profile B and other profiles that have chosen to follow Profile B.

In the IEEE 1301 family of standards, SU stands for Standard Unit, which is the smallest indivisible dimension in the card cage or subrack. These standard units are:

5 mm in the X direction of the subrack

25 mm in the Y direction of the subrack

25 mm in the Z direction of the subrack

By definition, a 12 SU x 12 SU subrack will fit in a cabinet cavity 300 mm high and 300 mm deep.

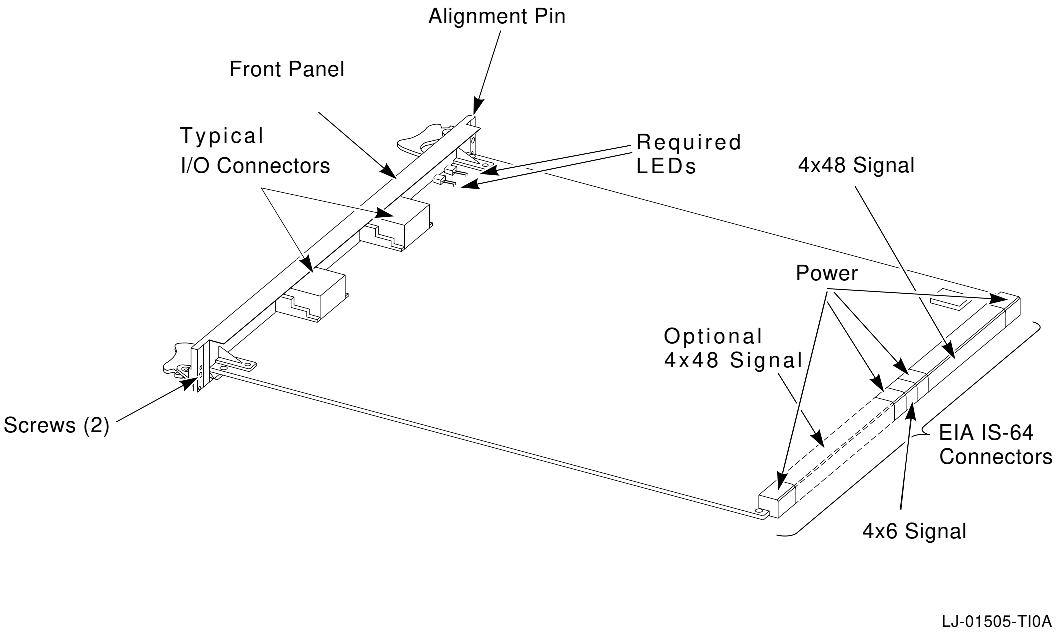
Intermodule separation planes are used as a reference to define clearances and to determine interferences between plug-in units. Plug-in units may occupy one or more subrack slots. The only hardware allowed to interfere with an intermodule separation plane is the hardware between slots of a multislot plug-in unit, and the electromagnetic interference (EMI) fingerstock on the front panel.

Figure 4–16, shows a Profile B single-slot, plug-in unit or module.

Continued on next page

Module Mechanical Specifications, Continued

Figure 4–16 Profile B Plug-In Unit or Module

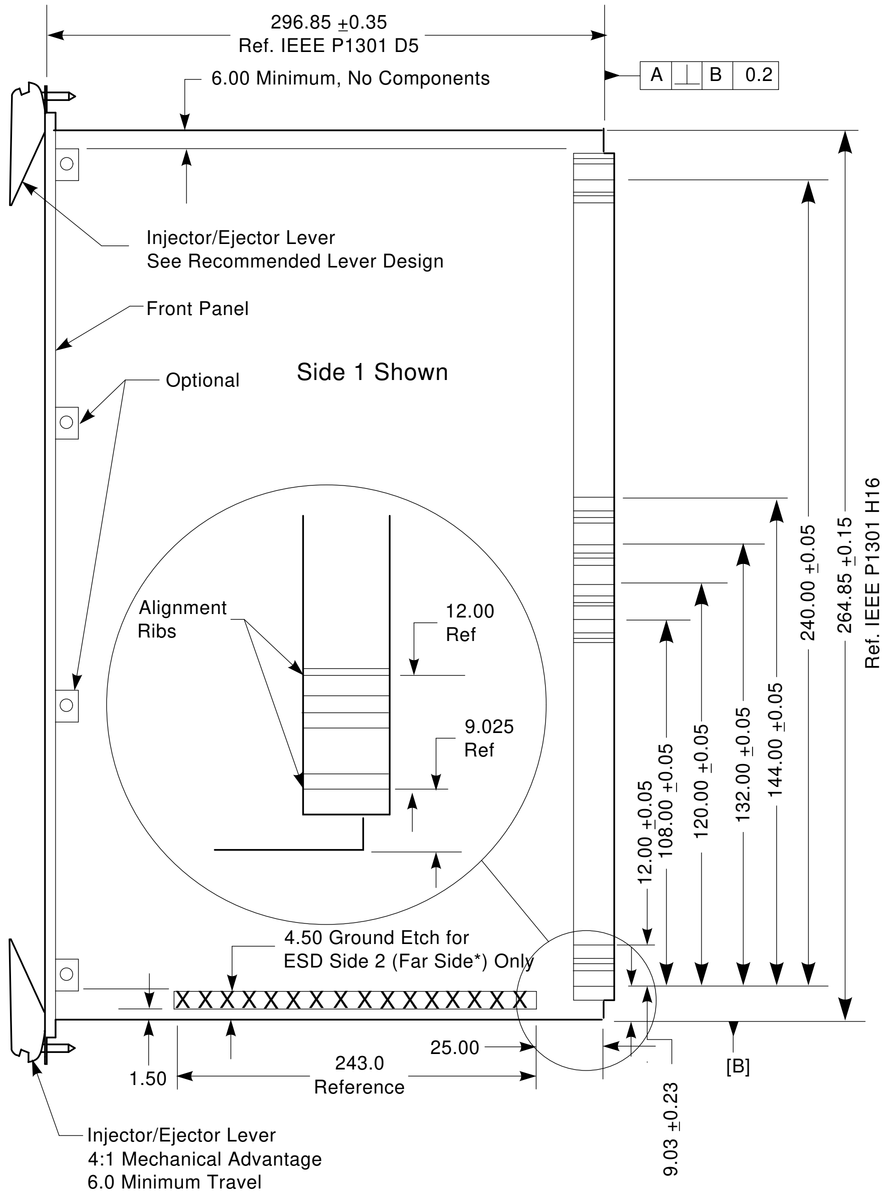


The connector set along the back edge of the module is the receptacle half of a two-part connector (specified in EIA IS-64) that mates with a pin header set mounted on the back panel. This connector set consists of both signal and power contact blocks that are mounted to the PWB via the PTH and nonplated hole pattern (shown in Figure 4–15). The connector is mechanically secured to the module by heat-stake pins from the connector to the module using nonplated holes.

Dimensions

The inspection dimensions that are used to guarantee intermateability between the plug-in unit and the subrack or card cage are shown in Figure 4–17.

Figure 4–17 Y-Z View of Single-Slot Plug-In Unit



* The words Far Side are used here to indicate that the FCC surface is not the surface that is visible.

LJ-01515-TI0

Module Component Height Restrictions

Overview

Module components are subject to skyline restrictions to prevent mechanical interferences between plug-in units (see Table 4–4 and Figure 4–18). The numbers apply to a single-slot plug-in unit. Allowable component heights may be increased in steps of 30 mm by using additional slots on the corresponding side of the PWB.

The component heights may be also be slightly increased on both sides of the PWB when the limits on bow are tightened, and on side 2, if the PWB thickness is reduced. Otherwise, violation of the skyline limits may lead to interferences between plug-in units unless additional slots are reserved (see Figure 4–18).

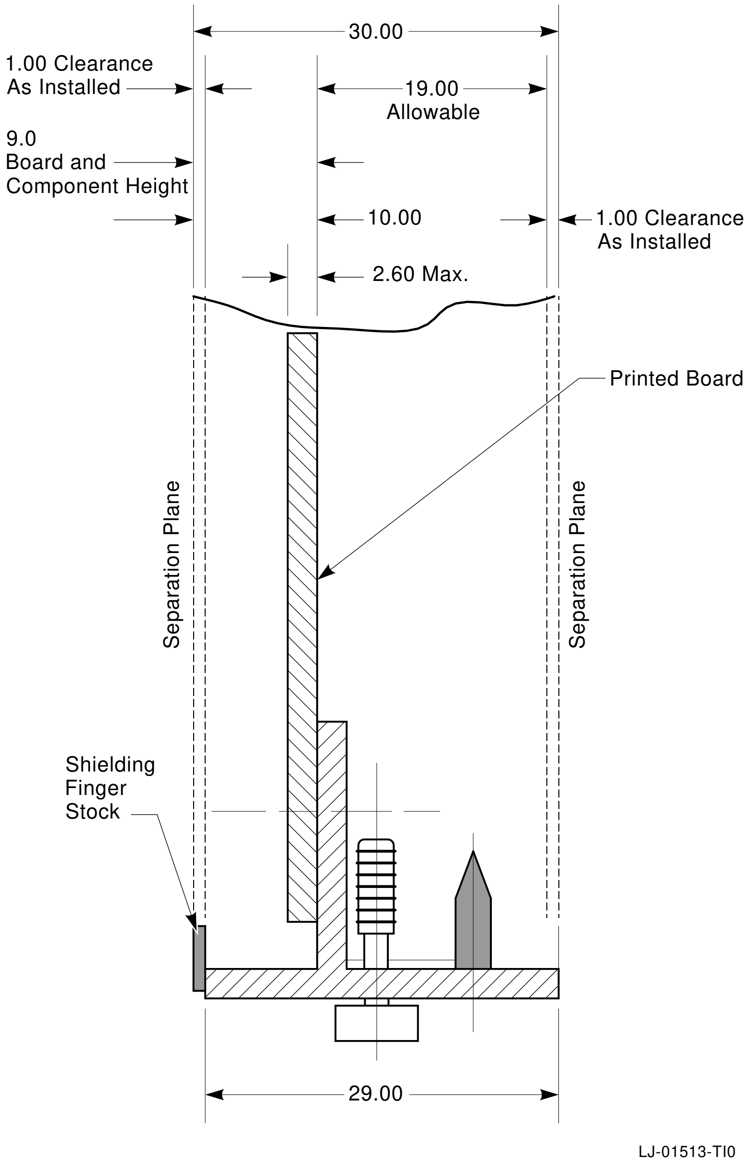
Continued on next page

Table 4–4 Skyline Limits for Single-Height Plug-In Unit

Distance between intermodule separation planes (module pitch)	30.0 mm
Distance between side 1 of the PWB and the intermodule separation plane on side 1 of the PWB	20.0 mm
Minimum clearance at the intermodule separation plane	1.0 mm
Available component height, module bow, and deflection, on side 1 of the module	19.0 mm
Typical specification for module bow (0.5% of the largest dimension)	1.4 mm
Allowance for dynamic deflection	0.6 mm
Available component height, with allowances for bow and deflection, on side 1 of the module	17.0 mm
Distance between side 1 of the PWB and the intermodule separation plane on side 2 of the PWB	10 mm
Maximum thickness of the PWB	2.57 mm
Minimum distance from side 2 of the PWB to the intermodule separation plane on side 2	7.43 mm
Minimum clearance at the intermodule separation plane	1.0 mm
Available component height, module bow, and deflection, for 2.57 mm PWB	6.43 mm
Typical specification for module bow (0.5% of the largest dimension)	1.4 mm
Allowance for dynamic deflection	0.6 mm
Maximum component height, with allowances for bow and deflection for 2.57 mm PWB	4.43 mm

Continued on next page

Figure 4–18 Intermodule Separation Planes, Clearances, and Maximum Component Heights



Module Front Panel

Overview

The Futurebus+ Profile B front panel is an integral part of the system. It serves:

- As a module front panel
- As a stiffening device for the module
- To contain/channel air flow
- To control EMI and satisfy EMC requirements
- To control ESD
- To support I/O cable connectors, switches, light-emitting diodes (LEDs), inject/eject levers
- To display labels, logos, etc.
- As a safety shield, to block access to the interior of the card cage
- To position module for proper insertion

Front and side views of the front panel, including dimensions, are shown in Figure 4–18, Figure 4–19, Figure 4–20, and Figure 4–21. Alternate designs that are compatible with Profile B appear in the IEEE 896.2 and 1301 standards.

Continued on next page

Electromagnetic Compatibility (EMC) Requirements

One of the functions of the front panel is to satisfy EMC requirements. These requirements are specified in a variety of government standards. Also, these standards evolve over time toward tighter limits, which often present severe design problems. If properly architected, however, the optimum solutions do not represent a serious cost impact.

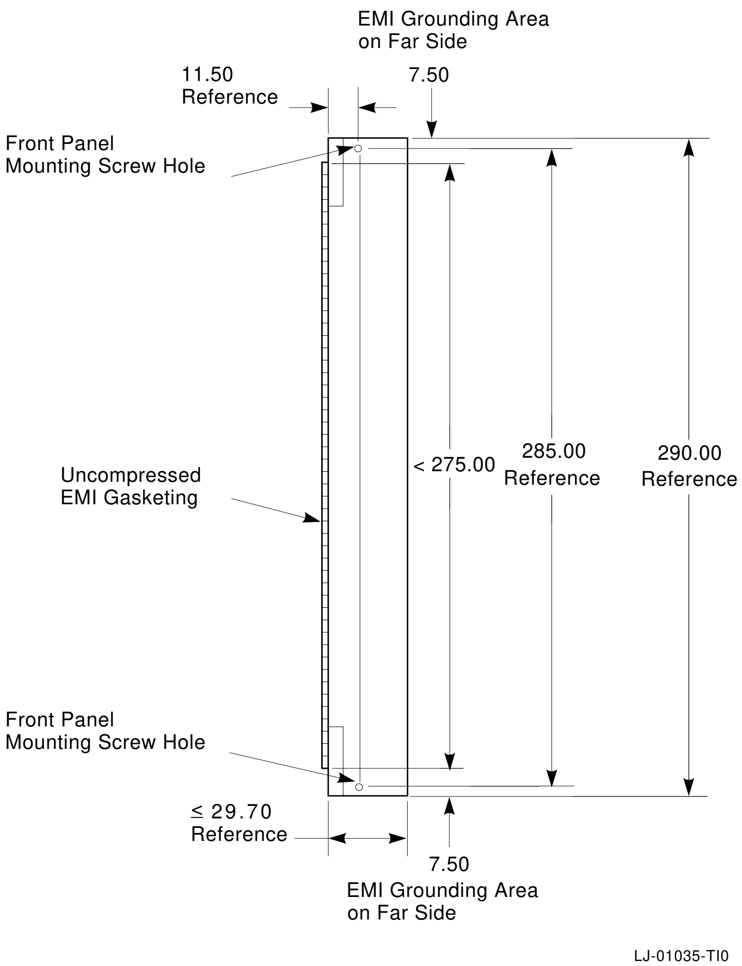
Profile B systems and options should both limit the internal sources of EMI generation and contain this generated EMI. This is accomplished by enveloping the card cage in a 6-sided Faraday shield, with the front panels of all the plug-in units constituting one of the 6 sides. Continuity of conductivity across the front panels is accomplished by using gasketing between the front panels. Conductivity to the other faces of the enclosure is accomplished by using spring clips at the upper and lower edges of the front panel. The base material may be either metal or plastic; however, plastic material must have a metalized finish. Regardless of the base material, acceptable shielding will be achieved if the surface resistivity does not exceed 0.1 Ω per square. Contact surfaces between front panels shall be galvanically compatible with nickel.

The width of the front panel shall be less than or equal to 29.7 mm (see Figure 4–19 and Figure 4–21). The remaining gap is taken up by the EMI RFI fingerstock that, in its uncompressed state, should expand to approximately 2 mm. To prevent galvanic action between the front panels of different vendors, the surface of the fingerstock should also be galvanically compatible with nickel.

Gasketing must also be provided at both the top and bottom surfaces that contact the subrack.

Continued on next page

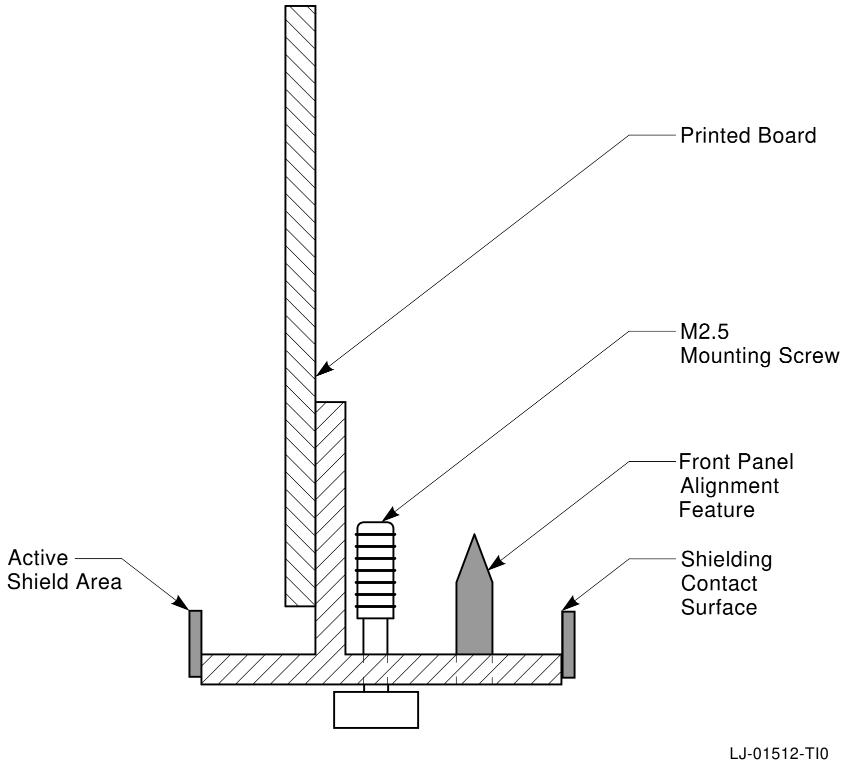
Figure 4–19 Y-Z Side View of Single-Slot Plug-In Unit



Continued on next page

Module Front Panel, Continued

Figure 4–20 Cross-Section of the Front Panel End



Gaps in the Faraday Shield

Gaps or holes in the Faraday shield, such as those needed for LEDs or switches that are larger than 3 mm in any direction, need to use the wave guide cutoff phenomenon to control high-frequence radiation. Table 4–5 shows the restrictions on front panel apertures.

Continued on next page

Table 4–5 Hole Dimensions for EMI RFI Compatibility

Max Hole Dimension	Min Depth of Hole	Closest Internal Source
3.0 mm	no limit	3.0 mm
7.5 mm	2.5 mm	7.5 mm
10.0 mm	10.0 mm	10.0 mm
Larger than 10.0 mm	Backshielding needed	1 hole diameter

ESD Requirements

Electrostatic discharges that occur directly to the front panel are required to use the card cage as the discharge return path. For this reason, the front panel is made of conductive material. The PWB is not intended to carry static discharges from the front panel.

Front Panel Attachment

A front panel attaches to the PWB along the front face of the board using 4 mounting tabs, 2 of which are optional. An EMI gasket attached to the left side of the front panel (side 2 of the module) fills the gap between the front panel and its left neighbor (see Figure 4–19).

Other components comprising the front panel include:

- Inject/Eject levers
- 2.5 mm Securing screws
- Locator pins
- Roll pins
- Identifying labeling

The lever, shown in Figure 4–21, provides approximately a 4:1 mechanical advantage. The alternate lever, shown in the IEEE 896.2 standard, Detail A, Side View of Front Panel Detail, offers a mechanical advantage of approximately 1:1.

Continued on next page

Figure 4–21 Detailed View of the Injector/Ejector Front Panel Mechanism

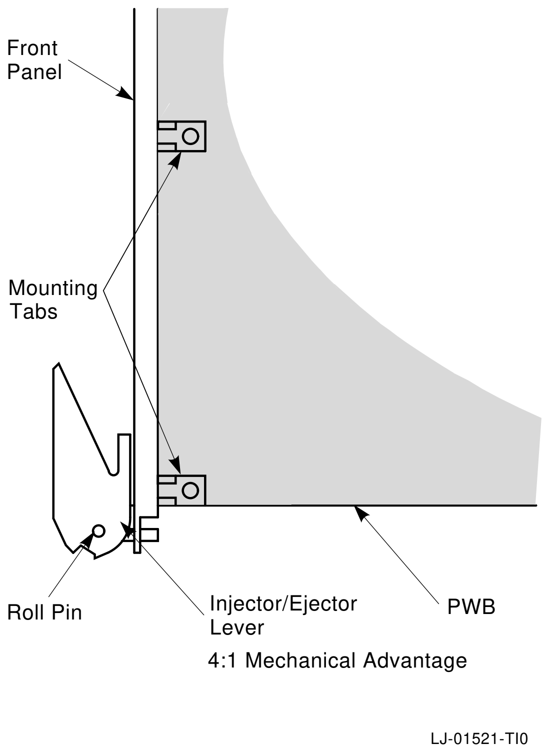


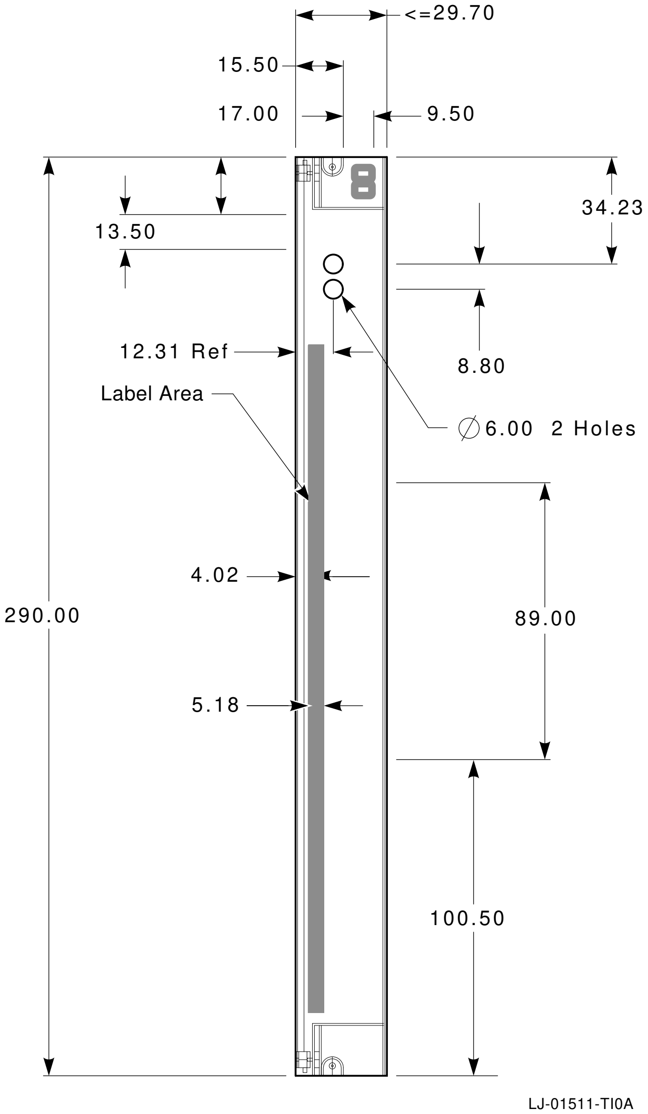
Figure 4–20 shows a cross-section of the front panel end.

Profile B Ergonomic Uniformity

The recommended specifications for providing ergonomic uniformity on Profile B front panels are shown in Figure 4–22.

Continued on next page

Figure 4–22 Recommended Locations for Labels, Logos, and Indicators



Module I/O Connections

I/O Signal Connections

All I/O connections must be made through the bulkhead/front panel area. Users may position I/O connectors anywhere along the defined user area of the front panel as shown in IEEE 896.2. (The front panel mounting hardware may require customization.)

I/O signal connectors attach to the front panel of the module to allow maximum flexibility with I/O signal connections, and because the concept of general purpose systems is based on the ability to freely mix Profile B modules within the Profile B system backplane. The front panel combines the functions of connector bulkhead, airflow control, and EMI shielding to permit compliance with international radiated emission standards. Modules may also implement I/O through connector E and out the backplane. However, modules shall also replicate that I/O through the front panel of the module. (Modules must not use any Profile B signal pins for I/O.)

I/O Cable Dress

External I/O cables are dressed downward along the face of the front panel. To enable the clearance of doors that may cover the front of the enclosure, cables must not project outwards more than 75 mm from the front face of the front panel.

Module LEDs

Physical Positions

Futurebus+ modules must incorporate, at a minimum, two LEDs:

- A green **RUN** LED
- A yellow **FAULT** LED

The relative position of these LEDs is recommended by the Futurebus+ mechanical specifications. In addition, the designer must take care to ensure that these LEDs are not an avenue of EMI/RFI leakage through the front panel. This is generally achieved by recessing the LEDs behind the Faraday shield according to the rules in Table 4–5.

Viewing Angle

These LEDs must be visible throughout a minimum solid viewing angle of 120°. A plastic light pipe may be necessary to achieve an acceptable viewing angle while still maintaining an adequate EMC shield.

LED Colors

The green LED shall have a typical wavelength of 565 mm. the yellow LED shall have a typical wavelength of 580 mm.

Module Keying

Overview

Keying of the module to the back panel is an option defined in the IEEE standards; it is not provided by the manufacturer. Digital systems do not implement keying; however, the systems do allow keying using one of the following two methods:

- Keying Tabs
- Keying Bridge

Keying Tabs

The keying tabs are applied to both the module receptacle and the pin header (as described in EIA IS-64 and shown in Figure 4–23). It should be noted, however, that this keying has limited effect on Profile B modules given the mechanical advantage that is supplied by the injector levers and the ability to crush the keying tabs. Studies show that only the key tab variations that have a full 1/2 width block, mated against a full 1/2 width block, have sufficient strength to effectively prevent a module from being inserted.

The keying tabs with smaller blocking areas are not recommended for backplane applications. By using only the two keys that survive the insertion forces generated by the insertion levers, and using them symmetrically in pairs to eliminate insertion torque and partial insertion, 2^{10} key combinations are left. This is more than sufficient for any user to key all products, however, not sufficient for the industry to manage all IEEE 1301.1 keying requirements.

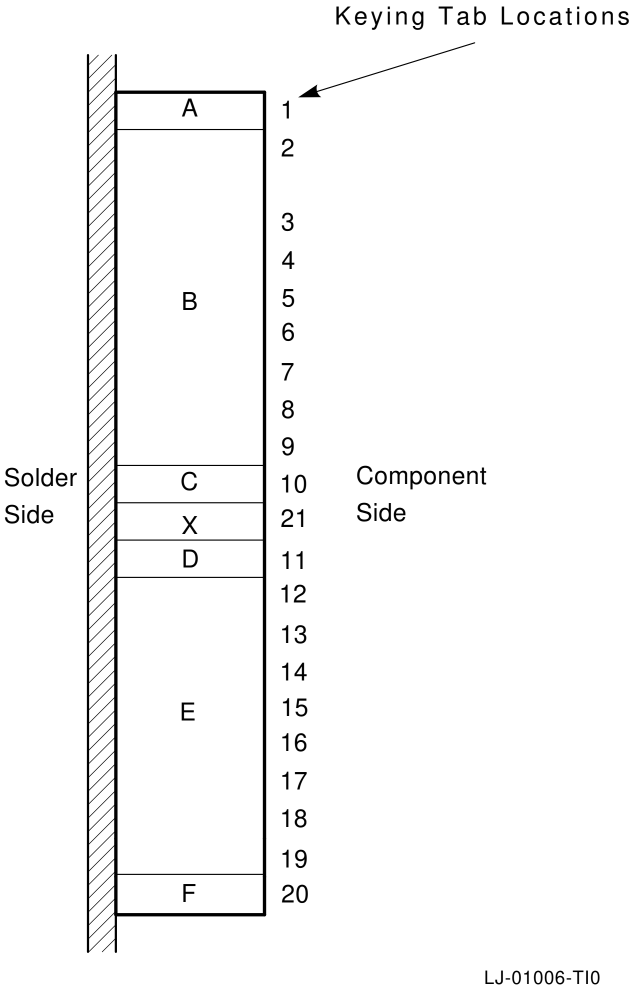
Keying Bridge

A keying bridge is affixed to the module, outboard of the connector set, that interfaces with a mating part on the back panel. This method requires the placement of holes in both the module and backplane to facilitate the mounting of this keying hardware. Digital systems provide these holes.

Continued on next page

Module Keying, Continued

Figure 4–23 Keying Tab Locations



Bus Connectors

Description of EIA IS-64 Connector

Futurebus+ Profile B modules require use of 2 mm EIA IS-64 connector system, which comes in signal and power versions. The EIA IS-64 connector consists of:

- Male backplane signal and power connectors
- Female module signal and power connectors

The EIA IS-64 connector family provides four rows of signal contacts with 2 mm pitch. The 2 mm pitch allows for 20 contacts per linear centimeter of connector ($\approx$50 contacts per linear inch of connector). In addition to signal pins on 2 mm centers, the EIA IS-64 connector family also provides power blades that are *triple wide*.

Connector Grounding Pattern

The connector grounding pattern is fixed according to IEEE 896.2. There is one ground reference pin for every two logic signal pins. The connector locations, type, and requirements are shown in Table 4–6.

Table 4–6 Connector Descriptions

Conn Loc	Type	Number of Pins	Requirement	Notes
A	power	6	Mandatory	
B	signal	48	Mandatory	Contains a 32/64 bit address/data bus.
C	power	6	Mandatory	
X	signal for central arbitration	6	Mandatory	Contains signals for central arbitration.
D	power	6	Mandatory	
E	signal	48	Optional	Refer to Multiple Futurebus+ Data Widths for more information.
F	power	6	Mandatory	

Optional Connectors

NOTE

All power connector sections are required. In modules or backplanes that do not implement 128-bit data or I/O PTH connector group "E", Section "E" of the signal connectors can be omitted.

Bus Connector and Pin Naming

Reference

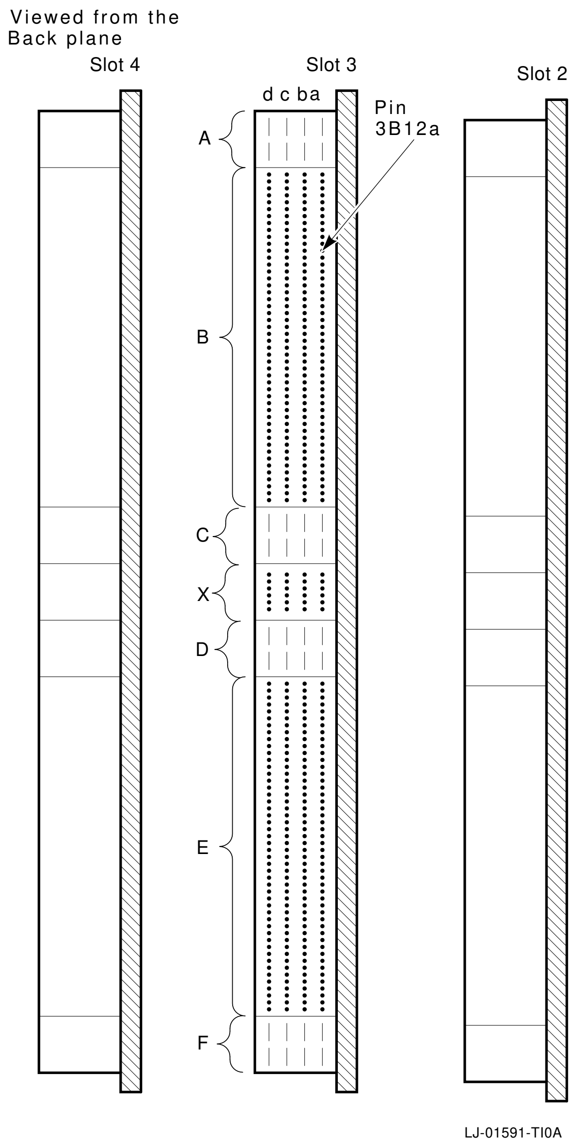
Connector pins are referenced by:

- Slot number of the module in the backplane (0 to 13 or 1 to 14)
- Connector group letter (A, B, C, X, D, E, or F)
- Pin row number (1 to 48)
- Pin column letter (a, b, c, or d)

For example, Figure 4–24 shows the location of pin 3B12a. This pin is located on the third module, connector group B, in the twelfth pin row, first column (the "a" column).

Continued on next page

Figure 4–24 Example of Connector Pin Naming



Futurebus+ Connector Blocks

Connector Blocks A, C, D, and F

The following figures show connector blocks A, C, D, and F.

Connector Block A

Figure 4–25 Backplane Connector Block A

	a	b	c	d	
1	+5V	+VBP	0V	+3.3V	1
2	+5V	0V	0V	+3.3V	2

LJ-01007-TI0

Connector Block C

Figure 4–26 Backplane Connector Block C

	a	b	c	d	
1	+5V	0V	0V	+3.3V	1
2	+5V	0V	+3.3V	+3.3V	2

LJ-01009-TI0

Connector Block D

Figure 4–27 Backplane Connector Block D

	a	b	c	d	
1	+5V	0V	+5V	+3.3V	1
2	+5V	0V	0V	+3.3V	2

LJ-01011-TI0

Continued on next page

Futurebus+ Connector Blocks, Continued

Connector Block F

Figure 4–28 Backplane Connector Block F

	a	b	c	d	
1	+5V	0V	+V	+V	1
2	+3.3V	+VBP	-V	-V	2

LJ-01013-TI0

Connector Blocks B, X, and E

The following figures show connector blocks B, X, and E.

Continued on next page

Connector Block B

Figure 4–29 Backplane Connector Block B

	a	b	c	d	
1	RE*	AI*	DS*	GA0*	1
2	GA1*	AK*	GND	AR*	2
3	AS*	GND	AQ*	AP*	3
4	GND	DI*	DK*	GND	4
5	GA2*	GA3*	GND	GA4*	5
6	ET*	GND	AC0*	AC1*	6
7	GND	CA0*	CA1*	GND	7
8	CA2*	ABP*	GND	AB0*	8
9	AB1*	GND	AB2*	AB3*	9
10	GND	AB4*	AB5*	GND	10
11	AB6*	AB7*	GND	CP*	11
12	CM0*	GND	CM1*	CM2*	12
13	GND	CM3*	CM4*	GND	13
14	CM5*	CM6*	GND	CM7*	14
15	SB0*	GND	SB1*	ST0*	15
16	GND	ST1*	ST2*	GND	16
17	ST3*	ST4*	GND	ST5*	17
18	ST6*	GND	ST7*	TG0*	18
19	GND	TG1*	TG2*	GND	19
20	TG3*	TG4*	GND	TG5*	20
21	TG6*	GND	TG7*	TP*	21
22	GND	AD0*	AD1*	GND	22
23	AD2*	AD3*	GND	AD4*	23
24	AD5*	GND	AD6*	AD7*	24
25	GND	BP0*	AD8*	GND	25
26	AD9*	AD10*	GND	AD11*	26
27	AD12*	GND	AD13*	AD14*	27
28	GND	AD15*	BP1*	GND	28
29	AD16*	AD17*	GND	AD18*	29
30	AD19*	GND	AD20*	AD21*	30
31	GND	AD22*	AD23*	GND	31
32	BP2*	AD24*	GND	AD25*	32
33	AD26*	GND	AD27*	AD28*	33
34	GND	AD29*	AD30*	GND	34
35	AD31*	BP3*	GND	AD32*	35
36	AD33*	GND	AD34*	AD35*	36
37	GND*	AD36*	AD37*	GND	37
38	AD38*	AD39*	GND	BP4*	38
39	AD40*	GND	AD41*	AD42*	39
40	GND	AD43*	AD44*	GND	40
41	AD45*	AD46*	GND	AD47*	41
42	BP5*	GND	AD48*	AD49*	42
43	GND	AD50*	AD51*	GND	43
44	AD52*	AD53*	GND	AD54*	44
45	AD55*	GND	BP6*	AD56*	45
46	GND	AD57*	AD58*	GND	46
47	AD59*	AD60*	GND	AD61*	47
48	AD62*	GND	AD63*	BP7*	48

LJ-01008-TI0

Continued on next page

Futurebus+ Connector Blocks, Continued

Connector Block X

Figure 4–30 Backplane Connector Block X

	a	b	c	d	
1	GND	RQ0*	RQ1*	GND	1
2	GR*	PE*	GND	Rsvd	2
3	Rsvd	GND	Rsvd	Rsvd	3
4	GND	Rsvd	Rsvd	GND	4
5	Rsvd	Rsvd	GND	Rsvd	5
6	Rsvd	GND	Rsvd	Rsvd	6

LJ-01010-TI0

NOTE:
Connector X pins 3a and 3c shall be bussed on the backplane.

Continued on next page

Connector Block E

Figure 4–31 Backplane Connector Block E

	a	b	c	d	
1	GND	D64*	D65*	GND	1
2	D66*	D67*	GND	D68*	2
3	D69*	GND	D70*	D71*	3
4	GND	BP8*	D72*	GND	4
5	D73*	D74*	GND	D75*	5
6	D76*	GND	D77*	D78*	6
7	GND	D79*	BP9*	GND	7
8	D80*	D81*	GND	D82*	8
9	D83*	GND	D84*	D85*	9
10	GND	D86*	D87*	GND	10
11	BP10*	D88*	GND	D89*	11
12	D90*	GND	D91*	D92*	12
13	GND	D93*	D94*	GND	13
14	D95*	BP11*	GND	D96*	14
15	D97*	GND	D98*	D99*	15
16	GND	D100*	D101*	GND	16
17	D102*	D103*	GND	BP12*	17
18	D104*	GND	D105*	D106*	18
19	GND	D107*	D108*	GND	19
20	D109*	D110*	GND	D111*	20
21	BP13*	GND	D112*	D113*	21
22	GND	D114*	D115*	GND	22
23	D116*	D117*	GND	D118*	23
24	D119*	GND	BP14*	D120*	24
25	GND	D121*	D122*	GND	25
26	D123*	D124*	GND	D125*	26
27	D126*	GND	D127*	BP15*	27
28	GND	GND	GND	GND	28
29					29
30					30
31					31
32					32
33					33
34					34
35					35
36					36
37					37
38					38
39					39
40					40
41					41
42					42
43					43
44					44
45					45
46					46
47					47
48					48

LJ-01012-TI0

Card Cage

Description

The card cage (referred to as subrack in IEEE standards) is a physical structure that contains the openings for plug-in modules. The card cage provides the guiding and control surfaces that allow insertion of the module within the capture range of the connector set, as well as control of the final seated position. In Digital systems, the card cage becomes an enclosed, 6-sided Faraday shield, with the front panels of all plug-in units constituting one side of the shield, and the remaining card cage assembly providing the other 5 sides.

The card cage (shown in Figure 4–32) consists of the following basic, functional components of card guides:

- Back panel assembly
- Top and bottom mounting rails
- Cage mounting brackets
- Side walls

Dimensions

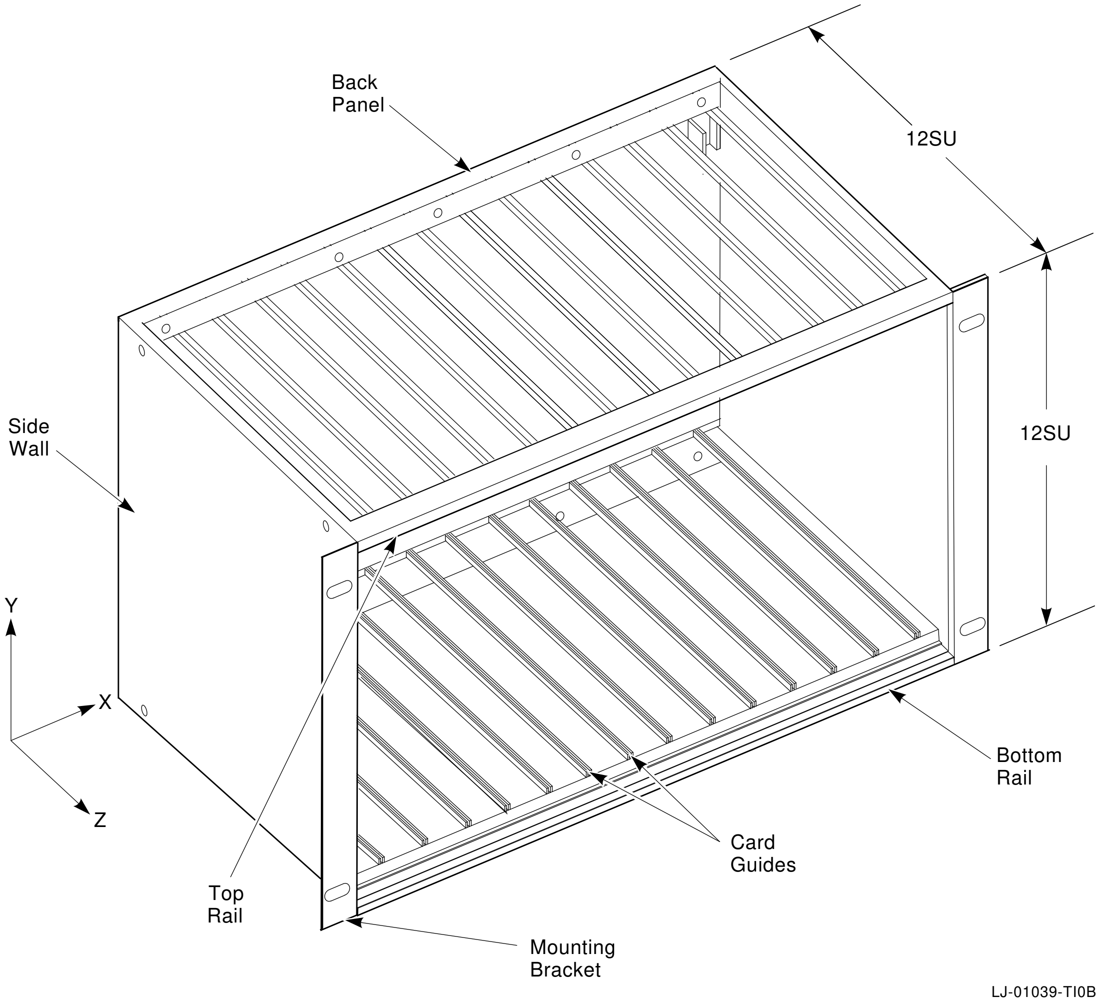
There are several dimensions that are of importance both in the design of the card cage and to the designer of a module that must interface to the card cage. Figure 4–33 shows the spacing between the card guides that accommodate the module size. The IEEE 1301 standard states that H5 dimension for a 12 SU space is 266.00 mm ±0.30 mm. Digital studies showed that conditions existed such that this spacing could yield interference; therefore, Digital uses the dimension 266.30 mm ±0.30 mm, as shown.

Continued on next page

Slot Pitch

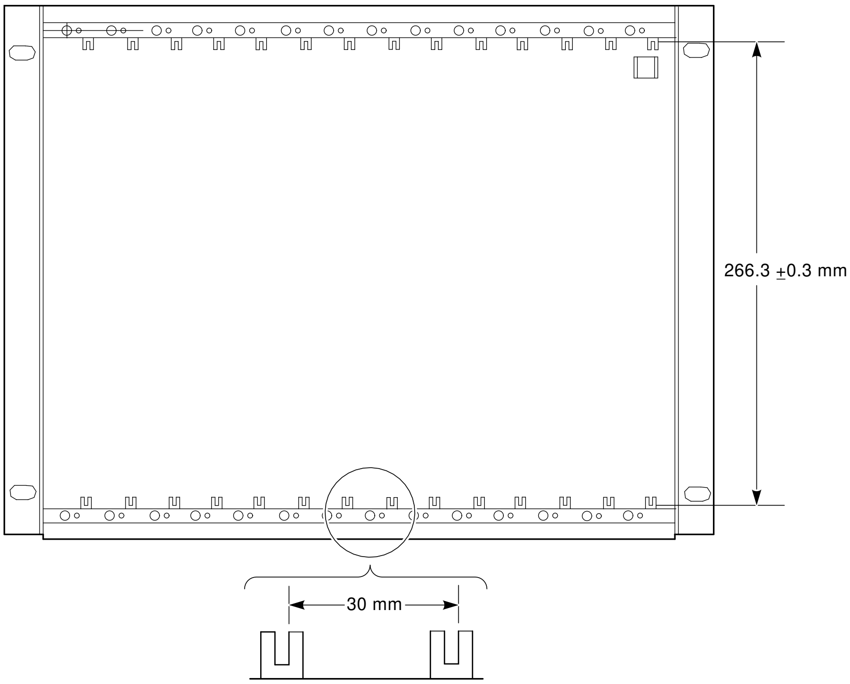
Futurebus+ Profile B modules are mounted on a 30 mm module-to-module pitch.

Figure 4–32 Card Cage



Continued on next page

Figure 4–33 Front View of the Card Cage



Side View

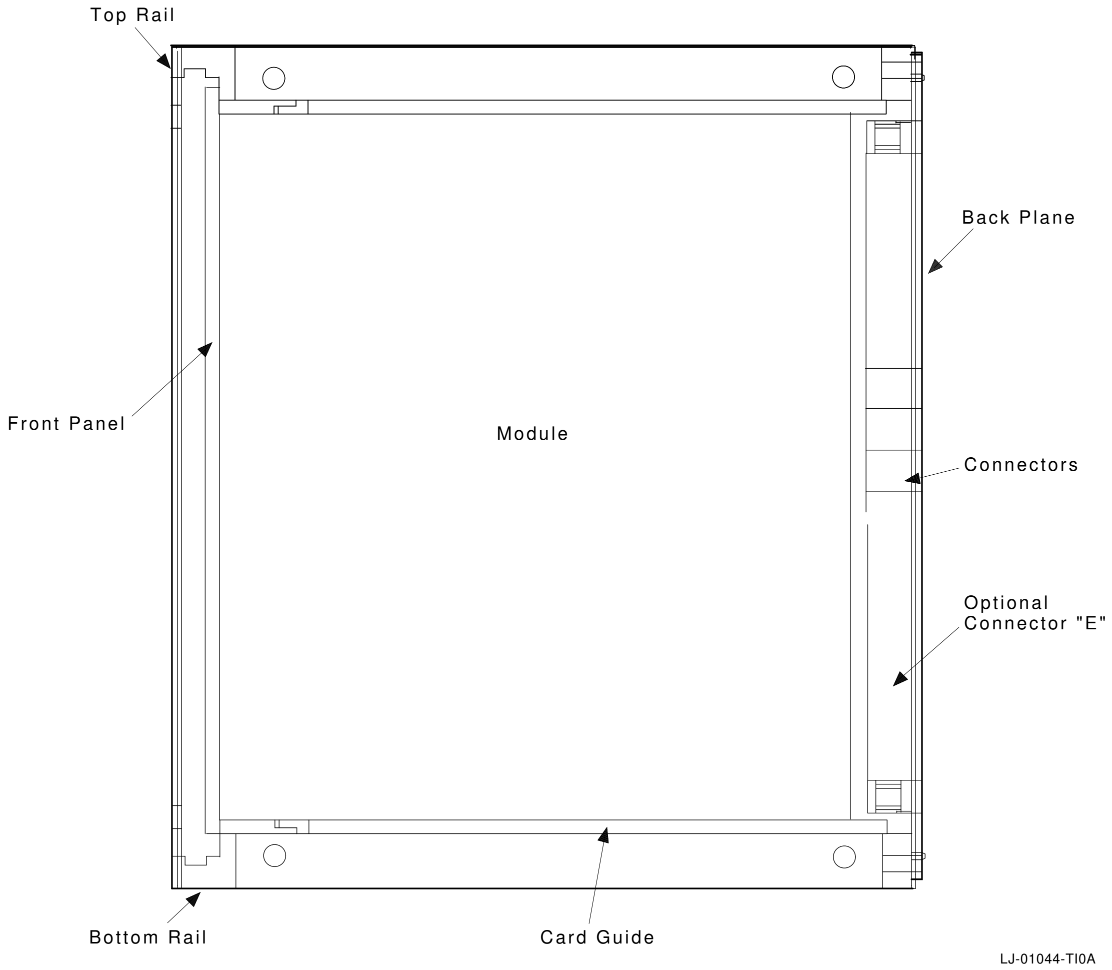
Figure 4–34 shows a side view through the card cage. This view identifies the following:

- Location of the backplane
- Front rails (top and bottom) and module assembly with front panel
- The correct spacing between the inside surface of the backplane connector and the outside card cage mating surface for the inside of the front panel

  The IEEE 896.2 standard states that this space is 297.35 mm ±1.1 mm. Digital studies showed that conditions existed such that this spacing could yield interference; therefore, Digital uses the dimension 297.85 mm ±0.8 mm (shown in Figure 4–34).

Continued on next page

Figure 4–34 Side View of the Card Cage



ESD Clip

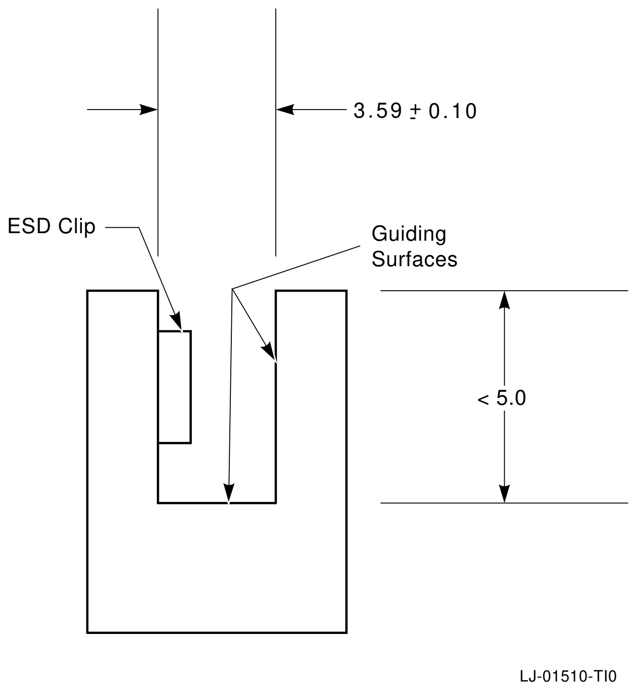
Figure 4–35 shows the card guide in more detail, indicating the location of the nickel-plated ESD clip and the card guide opening sizes. The ESD clip makes contact with the ESD etch on side 2 of the module.

Continued on next page

ESD Clip (continued)

As the module is being inserted, this ESD clip provides an early contact point to the module to draw off any static charge, and positions the module against the other side (side 1) of the card guide. Only the bottom card guide contains the ESD clip.

Figure 4–35 Card Guide Detail

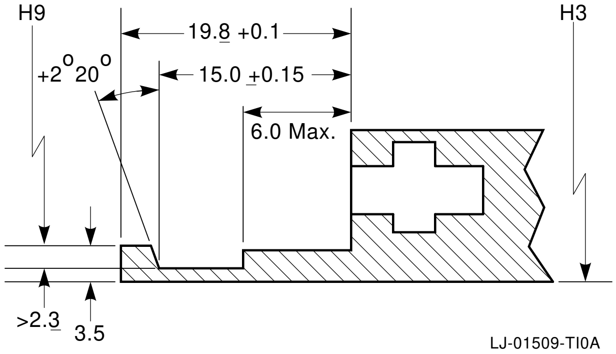


Top and Bottom Rail

Figure 4–36 shows a cross-section of a typical bottom (or top) rail defining the section that interfaces with the injector/ejector front panel mechanism.

Continued on next page

Figure 4–36 Top/Bottom Card Cage Rail

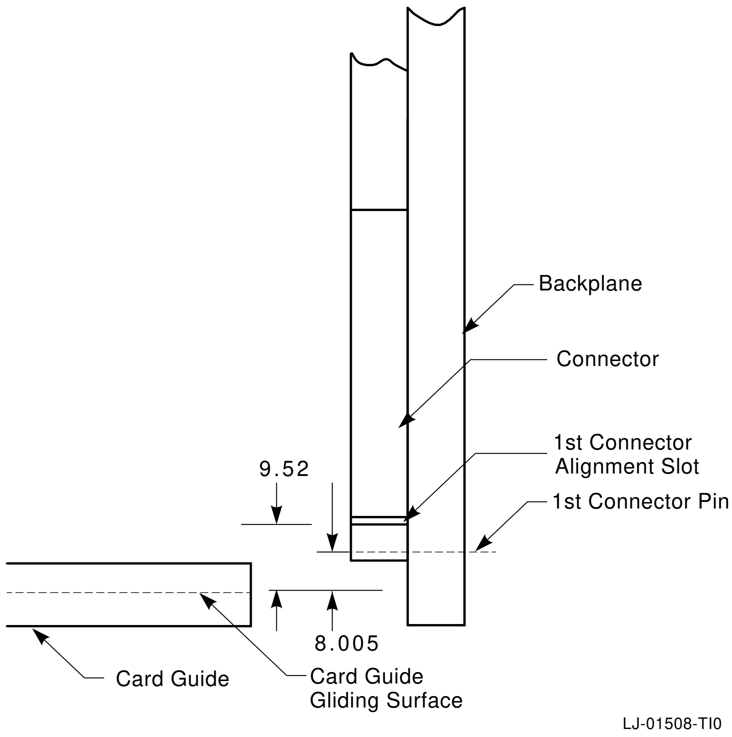


Mateability

Digital card cage designs provide the relationship of the pin header on the backplane to the height of the card guide controlling surface (see Figure 4–37 for the function dimension). This relationship ensures the module mateability in height with the connector above the bottom edge of the module (see Figure 4–38).

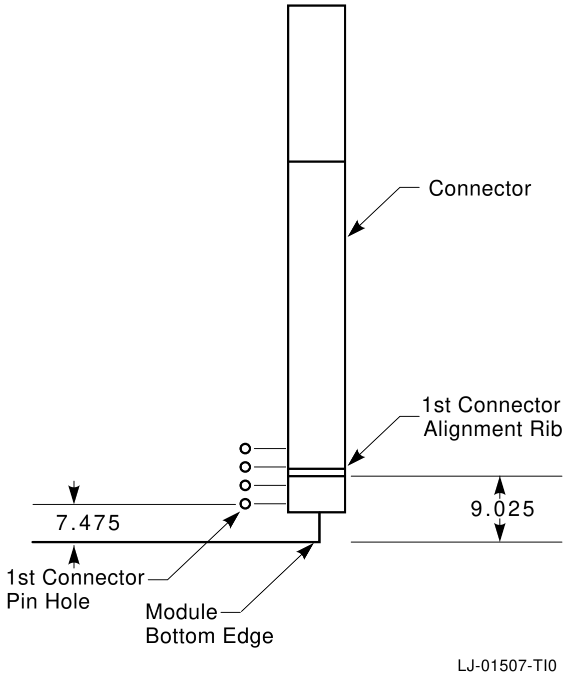
Continued on next page

Figure 4–37 Intermateability at Card Cage Level



Continued on next page

Figure 4–38 Intermateability at Module Level



Backplane Address/Data Widths

NOTE

Futurebus+ backplanes must bus 64-bit data. Futurebus+ backplanes may optionally bus 128-bit data. Similar to modules, backplanes can omit connector group "E" if only 32- or 64-bits are implemented and no modules will be routing I/O signals through connector group "E".

Backplane Layout Requirements

Signal Impedance

The backplane PWB must be designed to permit a controlled impedance. As shown in Table 4–1, this impedance must be in the range of 55 to 66 Ω for the bare backplane etch, including the effects of connector PTH.

Material

Similar to any backplane, Futurebus+ backplanes must be sufficiently rigid to allow module insertion and extraction with adequate pin wipe. The pin wipe budget, according to inspection dimensions of module assembly and card cage assembly for 6.5 mm pin, allows for 3.5 mm max to 1.55 min pin wipe.

Signal and Power Layers

The signal and power layers must be designed to accommodate the electrical characteristics described in Table 4–1.

Signal Switching

Rise time of the signal may be as fast as 1.5 ns 20% to 80%. This places difficult requirements on the design of the logic reference and termination voltage planes.

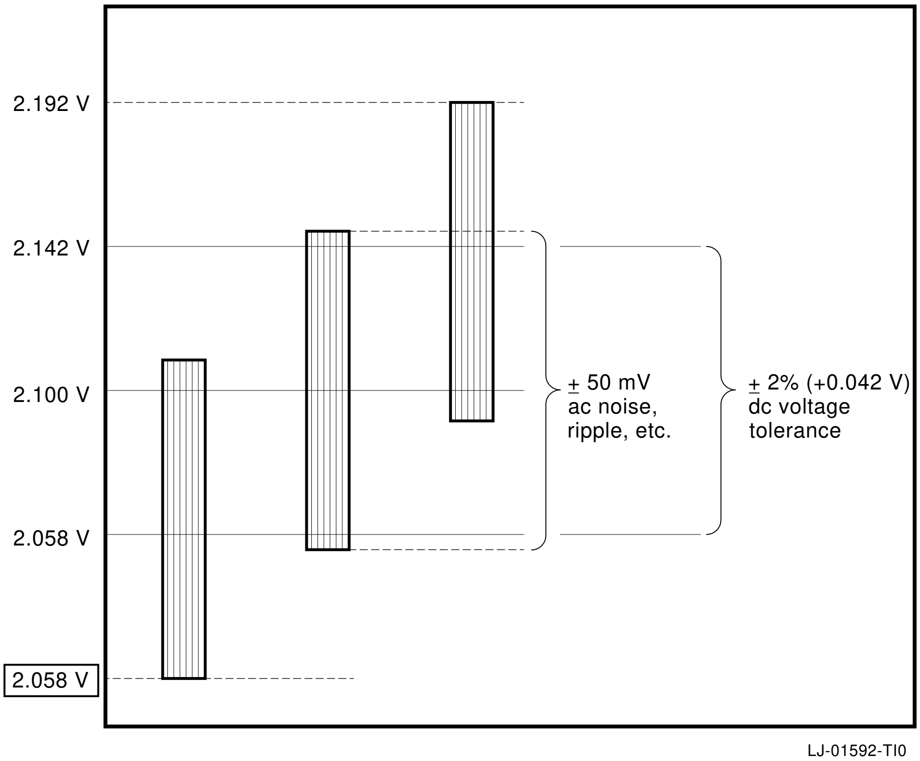
Backplane Signal Termination

Tolerance Requirements for +2.1 V

The BTL transceiver design and IEEE standards define a tight termination voltage tolerance of ±2% on the +2.1 V terminator voltage.

Because of the tight tolerance requirements, voltage losses in the terminator power system need to be minimized (within practical limits). Figure 4–39 shows the derivation of the tolerances for the signal terminations voltage.

Figure 4–39 Electrical Specifications



Continued on next page

Termination Resistor

All signals are terminated into 33 Ω connected to the +2.1 V termination voltage. Use 33 Ω ±1% surface mount devices for the termination resistors or a Thévenin equivalent circuit on each end.

Configuration Rules

NOTE
No more than 5 slots may be left empty between any two Futurebus+ modules.

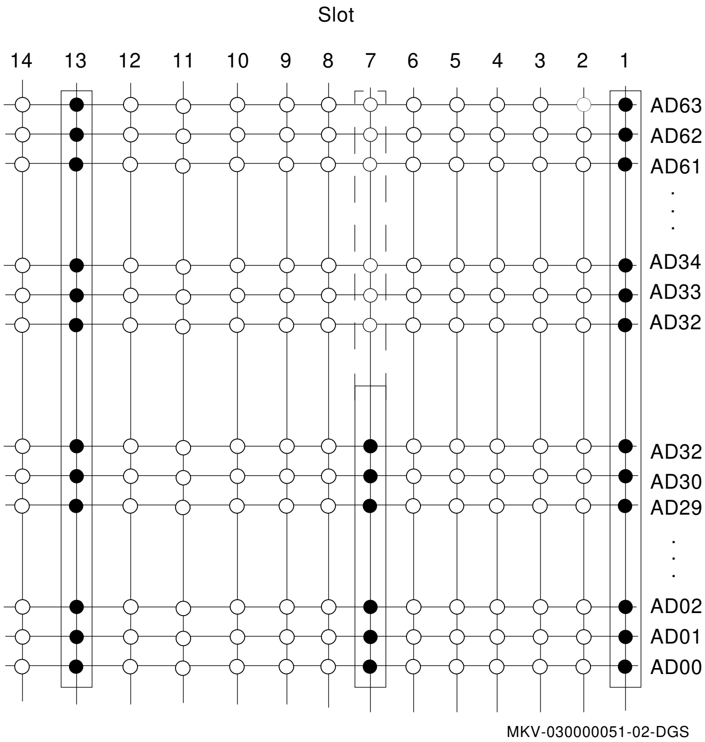
Details

This rule, while simple on the surface, becomes more complex in Futurebus+ systems that utilize modules of varying data width. Figure 4–40 shows an example of such a system. Even though there is a module every 6 slots, the module in slot 6 is only a D32 module and, therefore, does not connect to the AD[32..63] lines. There are 11 slots between the modules in slots 1 and 13, with respect to the modules with AD[32..63] lines.

This system has violated the "no more than 5 empty slots" rule and may not work.

Continued on next page

Figure 4–40 System With Modules of Different Data Widths



Power Supply (Required Outputs)

Power Supplies

Digital Profile B systems will supply +5.0 V, +3.3 V, and 48 V to all modules. In addition, Profile B systems must also supply or derive +2.1 V to the backplane for use in the termination of Futurebus+ signals on the backplane; +2.1 V is not supplied to the modules. Modules requiring other voltages may generate their specific voltage requirements locally. For more information, refer to the Backplane section in this chapter.

Power Supply (Output Sequencing)

Rise Time

Supply voltage waveform for all power supplies at turn-on shall be damped so there is no voltage overshoot greater than the maximum limit for that supply, as specified in IEEE 896.2, Voltage Variation section. Rise time shall be less than 200 ms from 10% to 90% of the nominal voltage under all specified loading conditions.

Power On Sequence

The power supply must sequence its outputs on as follows:

1. 48 V must turn on first.
2. Once 48 V reaches at least 38 V, +3.3 V may turn on.
3. The voltage on the +5.0 V rail shall not exceed the voltage on the +3.3 V rail until the 3.3 V rail reaches at least the +2.95 V.
4. The voltage on the 2.1 V termination rail must never exceed the voltage on the +5.0 V rail. In addition, the termination voltage must be within tolerance limits no later than 20 ms after the +5.0 V rail reaches 4.5 V.

Power Off Sequence

The voltage on the +5.0 V rail must never be more than 2.5 V more positive than the voltage on the +3.3 V rail.

Chapter 5
System Test

Introduction

Chapter Overview

This chapter describes the standardized approach for detecting faults. The chapter provides the guidelines to allow diagnostic testing to be independent of the instruction set architecture, system configuration, and operating system.

System test includes the following:

Structural test	Searches for manufacturing induced defects such as open or shorted wires.
Functional test	Verifies module functionality.
System diagnosis in the field	Verifies interaction problems between system components.

Goals

Introduction

To verify correct system operation and isolate failures, it is strongly recommended that each Futurebus+ module have the built-in capacity for high coverage self-testing, referred to as built-in self-test (BIST). BIST helps manage the cost and complexity of supporting systems that contain modules developed by different vendors on platforms running different operating systems and using different instruction set architectures.

The Futurebus+ standard provides definitions for many of the features of the test interface CSRs. There are also conventions that are recommended practices for BIST implementations. This chapter describes these standards and conventions, and distinguishes between those required by the standard and those that are recommended practices.

Test Management

The standard interface for managing diagnostic tests uses CSRs described in Chapter 3 in the IEEE 896.2 standard, and in Chapter 2 in this handbook. Testing is managed by the system configuration process and system-level diagnostics. Defining a CSR interface permits these processes to initiate diagnostics on all I/O modules, regardless of the functional character of the device.

Continued on next page

Futurebus+ Test Goals

The overall goals of system tests are as follows:

- It is strongly recommended that I/O devices have BIST diagnostics.
- BIST should provide a high degree of confidence that the operating system will boot on powerup.
- The boot data path should be testable without loading any additional software.
- The interface to I/O device diagnostics should be generic to enable an industry standard diagnostic monitor developed today to be capable of supporting current and future I/O devices.

BIST is strongly recommended for all Futurebus+ nodes or units in the boot path, and should be considered on all designs.

For example, on a system that supports booting from a SCSI device, all CPUs, memory, bus bridges, interconnects, and cables between the CPU and the boot device are part of the boot path and should be tested by high coverage BIST diagnostics. (A boot path device is any hardware component that may potentially be used to boot an operating system.)

Continued on next page

Goals, Continued

Strategies

The strategic goals of self-test and initialization diagnostics, in order of priority, are as follows:

1. Detection of failed nodes

   For simple nodes, this goal can lead directly to system repair by board replacement. For more complex nodes, this serves as a guide for selecting the correct repair agent or diagnostic procedure.

2. System-wide fault coverage

   This goal detects faults in system hardware before the execution of any system software. As the system software sizes the configuration, it can decide whether or not to use broken hardware.

3. Fault isolation of replaceable units within a module

   This goal seeks to eliminate the need for secondary diagnostic procedures (for example, loadable standalone diagnostics).

Continued on next page

Tactics

The tactics supporting the strategic goals, in order of priority, are as follows:

1. Minimal dependency on other Futurebus+ nodes

   While some dependency is unavoidable, the goal is fault isolation to a single module. If a system has only one fault, and the fault is in a given node, only that node should have *test failed* status set in the TEST_STATUS CSR. Some system-wide faults may cause fault-free Futurebus+ nodes to fail initialization or extended test (for example, faulty power supplies, missing terminators, broken memory).

2. Maximum coverage for elements that are likely to fail

   It is important to emphasize connectors, cables, device leads, receivers, and any high-priority failure modes as determined by a failure modes and effects analysis.

3. Coverage as far away (from the Futurebus+ backplane) as can be done without configuration dependencies

   BIST is specifically *not* restricted to testing the Futurebus+ interface; it is intended to test external buses, controllers, and devices that make up a module.

   External device testing requires some node-specific decisions about the meaning of test failed. If a node is the interface to a mandatory device and several optional devices, the test should fail if the mandatory device is missing or if any device has a fault.

4. Maximum coverage for gate failures (solid or stuck-at faults)

5. Fault isolation within a node

Continued on next page

Test Coverage Goals

The BIST diagnostics should be of high quality to enable all of the most likely failure modes and (most) other solid faults to be detected before they interfere with data integrity or otherwise fault-free system operation.

The combination of initialization self-tests and extended self-tests should have sufficient coverage to detect all faults that could interfere with booting the operating system.

The recommended test coverage is 100% of detectable faults, focusing on detection of stuck-at-0, stuck-at-1, and stuck-open faults on device signal pins (connectors, cables, device leads, drivers, receivers). Detection of greater than 80% of faults internal to very large scale integration (VLSI) devices is also recommended.

Defects may result in permanent or intermittent (temporary) faults. Although self-tests are designed to detect permanent faults, they are not precluded from detecting some temporary faults.

Built-In Self-Test

Overview

Each Futurebus+ should have diagnostic software and hardware built into each node, known as built-in self-test (BIST). BIST provides test and diagnostic capabilities that are independent of other nodes and instruction-set processors on the bus. A subset of BIST features is automatically activated upon system reset and directed reset of the node.

Component-Level BIST

The use of pseudo-random hardware self-test (linear feedback shift CSRs for pattern generation and signature compression) is recommended where appropriate. The use of an IEEE 1149.1 standard test access port and boundary scan is recommended for implementations that require boundary scan for interconnection testing.

Diagnostic Routine Definitions

There are specific areas that should be included in a built-in test package. The following classes of routines are invoked by a write to the standard test interface CSRs:

- Tests

Continued on next page

Diagnostic Routine Definitions (continued)

These are used to generate stimulus and compare the resulting response with an expected value. Tests are completely deterministic.

- Exercisers

  These are physical activity generators designed to make defects manifest themselves as errors. They are functional in nature and are aimed at nonlogical defects that manifest as pattern sensitivity, timing-related intermittent, or nonlogical device interaction errors. Exercisers are not entirely deterministic.

- Utilities

  These are diagnostic and support routines that are neither tests nor exercisers. Some examples of utilities include symptom collection routines, symptom analysis routines, configuration identification routines, firmware update routines, and disk formatters.

- Default tests

  These tests are a subset of the tests and exercisers selected so that their execution time is less than 10 seconds.

- Complete tests

  These tests are the complete set of nondestructive tests and exercisers selected without regard to any fixed time limit.

Continued on next page

Test Initiation

Tests are initiated by writing the test number into the TEST_START CSR.

Status Reporting

Test results are made available to other nodes on the Futurebus+ by recording status into the TEST_STATUS CSR. Digital systems will display a visual indicator of test results.

A watchdog timer will recognize when a test has hung. Expiration of the timer will cause the test to abort and timeout status will be reported in the TEST_STATUS CSR.

Diagnostic Hierarchy

Introduction

This section describes the type of diagnostic routine (test, exerciser, or utility) that should be included in each major diagnostic category. Test coverage should be provided in the first possible category. The combination of default initialization and default extended tests should cover enough of the logic that a primary or secondary bootstrap program will not encounter defects (90% or better stuck-at fault coverage).

The combination of all initialization and extended category tests should find all static defects that can be detected without manual intervention (95% or better stuck-at fault coverage). BIST should detect all static and dynamic defects on a module, such that booting the operating system is not a required manufacturing screen.

Diagnostic Categories

Diagnostics are divided into the following categories:

- Initialization tests

  These tests are self-contained on a node.

- Extended tests

  These tests require a memory buffer.

- System tests

  These tests require a memory buffer and another node.

- Manual intervention tests

  These tests require an operator.

Initialization Tests

Introduction

Initialization tests are autonomous in that they do not depend on the correct operation of other nodes in the system. Routines in the initialization category cannot use any resources beyond those local to the node or module under test. The time limit for the default set of initialization tests is 10 seconds. These tests should detect all permanent faults on a node that can be found without depending on the correct operation of other nodes in the system. (Futurebus+ drivers and receivers, some bus interface logic, and I/O port drivers and receivers are not tested.)

Initialization tests may be implemented as a collection of tests that are run in sequence by writing the RESET_START CSR. The tests may also be run separately by writing to the TEST_START CSR.

Some designs will have elements that are not fully testable in a short period of time (for example, large caches). Tests that take a long time to execute are not run by default.

Test Setup

No special setup is required to invoke the initialization tests.

NOTE
Invoking the initialization tests may cause the I/O module to terminate ongoing operations.

Continued on next page

Classes

The initialization tests can be grouped into the following classes:

- Initialization tests
  - Internal ROM checksum
  - Verification of all module-level data and control paths (all signal pins)
  - Local RAM, cache check, or both
  - BIST signature check (An Ethernet device may do external loopback on the network in the initialization category.)

Continued on next page

Classes (continued)

- Initialization exercisers
  - Nondestructive mass-storage device exerciser
  - Ethernet self-directed transfers
- Initialization utilities
  - Typically, no initialization utilities
- Default initialization tests
  - Less than 10 seconds run time on a defect-free module
  - High coverage of pin faults detectable in this category
- Complete set of initialization tests
  - No time limit

    Digital recommends using the TEST_STEP field in the TEST_STATUS CSR as a keep-alive indicator.
  - High coverage of pin faults, RAM cells, logic internal to VLSI, LSI, MSI, and SSI (ranging from very large to small scale integration) components, state machine logic, and so forth, detectable in this category
  - Complete testing of ROM and RAM structures

    This may take more than 10 seconds, and in some instances will not be part of the default initialization tests.

NOTE

Initialization tests do not perform transactions on the bus; therefore, they may be invoked on a running system without affecting other nodes.

Extended Tests

Introduction

Extended tests should detect all permanent faults that remain after initialization diagnostics on a module are executed, with the exception of certain system faults and logic that is untestable without operator intervention.

Tests in the extended category *must* initiate transactions on the bus. The object of the extended test category is to provide a guaranteed mechanism for detecting defects in Futurebus+ transceivers. The time limit for the default set of extended tests is 10 seconds. It is also a goal to use tests in this category to generate traffic on the Futurebus+.

Test Examples

The following are examples of extended tests:

- Transceiver tests

  The default set of extended tests are designed to verify that there are no opens or stuck-at-0 or stuck-at-1 faults on the Futurebus+ backplane transceiver logic (BTL) of the node. The tests should perform a series of write and read transactions to the 4 kbyte diagnostic buffer area in memory. This will verify that each individual command, data, handshaking, and control signal is asserted and released correctly.

- External tests

  These tests perform transactions on the Futurebus+ beyond those needed to verify the integrity of the bus transceivers. These tests may verify attached options such as mass storage devices and controllers.

Continued on next page

Test Setup

All nodes, with the possible exception of memory, should have extended tests to verify bus transceivers and other logic that are not testable at initialization time. Enable bits that will be set up before running these tests include, but are not limited to, bits in the following CSRs:

- LOGICAL_COMMON_CONTROL
- LOGICAL_MODULE_CONTROL
- BUSY_RETRY
- TRANSACTION_TIMEOUT

Bus transactions must be consistent with the control bits set in the LOGICAL_COMMON_CONTROL, LOGICAL_MODULE_CONTROL, and other Futurebus+ specific CSRs. For example, if the MASTER_ENABLE bit is not set, all extended tests shall fail because they are unable to initiate transactions on the bus. Extended tests may be implemented as a collection of tests that may be run in sequence by writing separately to the TEST_START CSR. These tests can only be implemented by designs that support the ARGUMENT CSRs.

Access to Memory

Transceiver tests depend upon the accessibility of a 4-kbyte block of physical memory. The memory space identified in the ARGUMENT CSRs is assumed to work and be accessible before running transceiver tests. The system configuration process will allocate a different block of space for every node on the bus. The default extended tests must take less than 10 seconds.

Continued on next page

Classes

The classes of routines and what the routines test are as follows:

- Extended tests
  - — Simple test of the transceivers
  - — Functional DMA test

    Unit-specific format DMA control data structures should be written into memory. The unit under test should then use that data structure to control a block transfer operation.
- Extended exercisers
  - — Pseudo-random sequence of transactions to the scratch buffer
  - — Variable data, address, and transaction types should be used.
- Extended utilities
  - — Typically, no extended utilities
- Default extended tests
  - — Less than 10 seconds run time on a defect-free module
  - — High ($\geq$90% stuck-at fault) coverage of pin faults detectable in this category

Continued on next page

Classes (continued)

- Complete set of extended tests
  - No time limit

    Digital recommends using the TEST_STEP field increment as a keep-alive indicator.
  - High ($\geq$95% stuck-at fault) coverage of pin faults, RAM cells, logic internal to VLSI, LSI, MSI, and SSI components, state machine logic, and so forth, detectable in this category

Attached Option Tests

If attached options have resident diagnostics, appropriate writes to the standard test interface specifying the extended category should cause those tests to run.

System Tests

Introduction

System tests are the option of the system designer. Some examples include verifying bus arbitration, global clock functions, message passing, or cache coherence in shared memory systems.

This type of testing requires some kind of memory management, and is dependent upon system configuration, diagnostic monitor software, and the test program. Enable bits that will be set up before running these tests include (but are not limited to) bits in the LOGICAL_COMMON_CONTROL, LOGICAL_MODULE_CONTROL, and TRANSACTION_TIMEOUT CSRs.

System tests should use the ARGUMENT CSRs to identify a block of memory that may be shared with other units under test for device interaction testing.

Optional System Test Conventions

The system test category is an option that the system designer may interpret in different (nonstandard) ways. The interpretation of data passed in the TEST_STEP field or in the ARGUMENT CSRs in system test is not specified in the IEEE 896.2 standard.

In the system test category, the TEST_STEP and TEST_STATUS CSRs may be used as an ASCII character port. Designs that have a resident diagnostic monitor or console may define a convention where a portion of the TEST_STEP field in the TEST_START CSR specifies that this is an ASCII character. The unit may echo transmit characters in the other direction through the TEST_STEP field in the TEST_STATUS CSR. This has been discussed, and is not prohibited by the standard; however, use of this technique requires a device-specific diagnostic control program.

Stressing the Bus

System tests may be used to stress the bus by generating a dense sequence of transactions that consume as much of the available bus bandwidth as possible.

Continued on next page

Classes

The following tests and exercisers apply to Profile B:

- System tests
  - Routines to support qualification and conformance testing
  - Force and verify detection of bad parity on the bus (requires cooperation of another module)

    This cannot be done in other test categories; other nodes would detect bad parity on the bus while this test is running
- System exercisers
  - Routines to test arbitration and other aspects of interaction between multiple modules
- System utilities
  - Routines to dump buffers (see the ‘‘Dump Buffers’’ section of this chapter).
- Default system tests
  - Dump buffers utility (Digital recommendation)
- Complete set of system tests

Manual Intervention Tests

Introduction

Manual intervention tests are optional. They require some action on the part of the operator, such as mounting scratch media, connecting an I/O loopback connector, or manually setting loopback on an external modem. Digital recommends that any loopback connector that may be required to diagnose a module in the field be included in the package with the module when it is installed. Thus, field technicians do not need to carry all possible varieties of connectors for on-site service.

Manual intervention tests may include verification of time-out counters, parity error detection, or other tests that require manually forcing an error.

Continued on next page

Manual Intervention Tests, Continued

Classes

The classes of routines and what these routines test are as follows:

- Manual intervention tests
  - Communication device external loopback
  - Mass storage device read/write tests
  - Live insertion tests
  - Power fail tests
- Manual intervention exercisers
  - Unit-dependent
- Manual intervention utilities
  - Disk formatters
  - Firmware load/update
  - Module nonvolatile memory access
- Default manual intervention tests
  - Typically, no default manual intervention tests
- Complete set of manual intervention tests
  - Typically, no complete set of manual intervention tests

ARGUMENT CSRs

Introduction

The ARGUMENT CSRs are used to pass addresses to extended or system category diagnostic tests. The use of these CSRs is specified in the extended tests category only. Other test categories may use these CSRs for implementation-dependent applications.

A system configuration process or a test control node should reserve an aligned block of memory for each node under test. A pointer should be provided to the block of memory by writing the starting address of the block into the ARGUMENT CSRs before invoking an extended test or an extended test sequence.

Writing the ARGUMENT CSRs

A write of a value to the ARGUMENT CSRs has the following effect on diagnostics:

- Initialization tests

  No effect
- Extended Tests

  Starting address of a 4-kbyte block of memory specified for extended self-test
- Manual intervention tests

  Vendor-defined
- System test

  Implementation-specific (System test may also define the starting address of a 4-kbyte block of memory for system and device interaction test.)

Reading the ARGUMENT CSRs

Read values may include supplemental information provided upon detection of an error.

Continued on next page

ARGUMENT CSRs, Continued

ARGUMENT CSRs and Initialization Tests

The default initialization test sequence will run automatically on powerup or when a system reset is signaled over the RE* line. The ARGUMENT CSRs shall not be used by tests in the initialization test category because these tests are run as a side effect of a system reset. In that environment, there is no opportunity for a test manager or system configuration process to load the contents of the CSRs.

Because of this, a test that runs in the initialization category with default parameters may have an alias test number in the system test category. When invoked in the system test category, such a test may use parameters passed in the ARGUMENT CSRs as input or output data.

ARGUMENT CSRs and System Test

By convention, the default system category test that dumps system error information shall be a utility that uses the same format for the ARGUMENT CSRs as is specified for the extended test category (that is, as a pointer to a memory buffer).

Aside from conformance test support, there is no standard or convention recommended or specified for other tests in the system test category.

Remote Tests

Introduction

Any node on the bus may become master by arbitrating for the bus. Since all diagnostic functions may be initiated by CSR access over the bus, any node may serve as an access port for remote diagnostics. It is up to the system designer to implement a remote diagnostic capability according to the requirements of the system.

System designers may choose to include the remote diagnostic port on a processor, a local area network device, any communication interface module in a system, or may choose not to include a remote access port.

Dump Buffer Utility

Introduction

There may be an I/O module resident firmware utility that has the function to dump a buffer or error log into memory. The buffer or error log may contain the results of the last failing diagnostic test, a machine check stack, contents of nonvolatile configuration data, or whatever a particular unit implementation needs to dump.

This utility is the method that Digital consoles and diagnostic monitors will use to access extended error information.

Invocation and Results

The utility will be invoked by running the default diagnostic test in the system test category (TEST_STEP = 0x0000). The utility will write to a (4-kbyte) buffer identified by the address passed in the ARGUMENT CSRs. The least significant two bytes in the first four bytes of the buffer contain the buffer size in bytes. The most significant two bytes contain a data type argument that specifies the format a utility analyzing or printing the buffer should use (for example, 0 = no format specified, 1 = ASCII, 2 = HEX). All other encodings are vendor-dependent.

A buffer or error log may consist of more than one 4-kbyte block of information. Specification of multiple block buffers is also vendor-dependent; however, it is recommended that each block start with a length field and type field. (It is not necessary to define a set of identifiers for different blocks of information, as these may be vendor-defined.) The only function the utility will perform is to dump the buffer into memory.

Futurebus+ Firmware Update Utility

Introduction

Any Futurebus+ device that contains loadable firmware should support local firmware update. The system integrator will provide a Futurebus+ Firmware Update Utility (FFUU) that will enable a user to update the firmware on any Futurebus+ adapter. The FFUU will provide a generic adapter update routine that will update any Futurebus+ adapter that implements the generic update strategy. The FFUU is also designed to enable each adapter group to write an adapter-specific adapter update routine to run in place of the generic routine.

Futurebus+ Generic Update Strategy

All adapters that implement the generic update strategy will allow for firmware updates using the following sequence of operations that are performed by the FFUU:

1. Load the adapter firmware update image into system memory at address X.
2. Reset the target device.
3. Load the target device ARGUMENT CSRs with address X.
4. Invoke device resident FFUU by writing the TEST_START CSR with CAT field = system, and TEST_STEP field = 0xC0DE.
5. Monitor the TEST_STATUS CSR. When the update is complete, reset the device and report the status to the user.

When the firmware update is complete, all adapters shall write TEST_STATUS with the completion status. Firmware update completion status messages should be standard for all Digital Futurebus+ adapters.

The firmware update error status codes are shown in the Field Replaceable Unit (FRU) Code field of Table 5–1.

Continued on next page

Table 5–1 Firmware Update Error Status Codes

FRU Code	Condition	Description
00	Success	File was loaded and verified.
01	Failure	Bad CRC. File to be loaded is corrupted.
02	Failure	Bad version. Incompatible version.
03	Failure	Bad device. Wrong device.
04	Failure	Timeout. Timed-out loading file.
05	Failure	Hardware failure on device. Code update failed.
06	Failure	Verify failed. Image was loaded and the subsequent verify pass failed.

Conformance Test Support

Introduction

Logical verification of slave state machines is relatively easy. The conformance test initiator may act as master and initiate a transaction and introduce any combination of status or error conditions. The master is the frame of reference.

To test for logical conformance of a master state machine and cause the unit to initiate specific transactions, a conformance test requires unit-specific knowledge of the unit under test. A recommended practice that will help verify conformance is the development of diagnostic tests that are designed with conformance test in mind.

Predefined Conformance Tests

To facilitate conformance testing and make it unit architecture independent, a series of built-in ROM-based tests have been defined that allow a Futurebus+ master to force a node to issue transactions on the Futurebus+.

These tests are specified so that a node can issue Futurebus+ transactions without having any unit initialization or software drivers. The tests require the implementation under test (IUT) to read and write system memory. The memory addresses will change depending on the conformance parameter being tested and the testing environment. The addresses must be passed to the IUT and be able to start on any boundary. These requirements mean the tests must be in the system test category.

Detailed Test Description

System test numbers 256 to 511 (0x0100 to 0x01FF) have been reserved for conformance purposes. A large block has been used to facilitate decoding and allow for future extensions. Five conformance tests have been defined. The test numbers and a brief description are listed in Table 5–2.

Each test is described in detail in the sections after the table.

Continued on next page

Table 5–2 Conformance Tests

Test Decimal	Hexadecimal	Description
256	0x0100	Initiate a Read Unlocked.
257	0x0101	Initiate a Read Partial.
258	0x0102	Initiate a Write Unlocked.
259	0x0103	Initiate a Write Partial.
260	0x0104	Initiate a Read Unlocked followed by a Write Unlocked of the data that was read.

NOTE
Modules that never issue partial transactions are not required to implement tests 257 and 259.

While these are called tests, they do not check for or log incorrect data or check for successful completion. The intent is for another Futurebus+ master to configure the node CSRs, pass it an address, start a conformance test, assert various error conditions or status conditions (both legal and illegal), and then check the behavior of the IUT and error CSRs for proper settings. After the test information has been logged, the IUT is reset and additional conformance tests may be performed. The IUT hardware is required to follow the proper protocol for status and error handling.

Continued on next page

Test Setup Requirements

Each test requires a general configuration setup. Table 5–3 lists the CSRs that will be set up in the IUT by the testing master and used by the IUT in determining transaction parameters (not necessarily in this order).

Table 5–3 CSRs in the IUT

NODE_IDS	This CSR sets up the bus address for the node for decoding CSRs and Read Response transactions.
SPLIT_TIMEOUT	This CSR sets the value for a split timeout error.
ARGUMENT_ (LO, HI)	These CSRs are used by the IUT as the source or destination address of the transaction.
TEST_START	This CSR sets up the test category, test number, and test operation.
LOGICAL_COMMON_CONTROL	This CSR sets up split enable (if implemented).
LOGICAL_MODULE_CONTROL	This CSR sets up the data length, data width, and address width.
BUS PROPAGATION	This CSR sets the glitch filter delays.
TRANSACTION_TIMEOUT	This CSR sets up the transaction timeout.
BUSY_RETRY_COUNTER	This CSR sets up the number of retries to a busy response.
BUSY_RETRY_DELAY	This CSR sets up the delay between the retried transactions.

Continued on next page

Test Setup Requirements (continued)

For example, if the IUT is configured for a data length of 0b011, a data width of D32, and an address of 0x00001004, the IUT would issue a transaction containing a connection phase address of 0x00001004 and three data beats. If the address passed to the IUT was 0x00001000, the IUT would issue a connection phase address of 0x00001000 and four data beats.

The IUT will use the configured data length, data width, and address width in performing each test. If the LOOP bit is set in the TOPS field, the module continually repeats the test until reset (using multiple transactions per tenure if the module supports this feature).

Read Unlocked (Test 256)

This test checks the ability of the IUT to properly issue Read Unlocked transactions under various conditions. The memory or CSR address will be passed to the IUT in the ARGUMENT_LO and ARGUMENT_HI CSR. The IUT will issue the Read Unlocked transaction to that address. The address will be minimum longword aligned (for D32) or quadword aligned (for D64).

Read Partial (Test 257)

This test checks the ability of the IUT to properly issue Read Partial transactions under various conditions. The memory or CSR address will be passed to the IUT in the ARGUMENT_LO and ARGUMENT_HI CSRs. The IUT will issue the Read Partial transaction to that address. The IUT will have to properly set the lane disable bits depending on the address passed to it. The address can be aligned to any byte address.

For example, assume that the IUT is given a data length of 1, a data width of D32, an address width of A32, and is passed address 0x00001001. The IUT should issue a Read Partial using address 0x00001000 (or 0x00001001 because lower significant address bits are user-defined) and enable byte lanes 3, 2, and 1. Byte lane 0 should be disabled.

Continued on next page

Write Unlocked (Test 258)

This test checks the ability of the IUT to properly issue Write Unlocked transactions under various conditions. The memory or CSR address will be passed to the IUT in the ARGUMENT_LO and ARGUMENT_HI CSRs. The IUT will issue the Write Unlocked transaction to that address. The address will be a minimum of 4-byte aligned (for D32) or 8-byte aligned (for D64). The data pattern written is as follows:

```
A5A50000            for D32
A5A50000A5A51111    for D64
```

For multiple data beats, this pattern is repeated for each beat.

Write Partial (Test 259)

This test checks the ability of the IUT to properly issue Write Partial transactions under various conditions. The memory or CSR address will be passed to the IUT in the ARGUMENT_LO and ARGUMENT_HI CSRs. The IUT will issue the Write Partial transaction to that address. The IUT will have to properly set the lane disable bits depending on the address passed to it. The address can be aligned to any byte address.

For example, assume the IUT is given a data length of 1, a data width of D32, an address width of A32, and is passed address 0x00001001. The IUT should issue Write Partial using address 0x00001000 (or 0x00001001 because lower significant address bits are user-defined) and enable byte lanes 3, 2, and 1. Byte lane 0 should be disabled.

The data pattern written is as follows. The disabled byte lanes can be driven to either state, however, the parity of these lanes must be correct.

```
A5A50000            for D32
A5A50000A5A51111    for D64
```

For multiple data beats, this pattern is repeated for each beat.

Continued on next page

Read Unlocked Followed By Write Unlocked (Test 260)

This test checks the ability of the IUT to correctly handle error conditions where data is required to be ignored and also that the error CSRs are properly set under multiple error conditions. The memory or CSR address will be passed to the IUT in the ARGUMENT_LO and ARGUMENT_HI CSRs. The IUT will issue a Read Unlocked transaction to that address followed by a Write Unlocked (using the data read as the write data) to the same address.

Test Number Assignment Conventions

Recommended Practice

One test number can define four different tests (a different test for each category). For example, test 20 may denote a RAM test in the initialization category, a host memory DMA test in the extended test category, and something entirely different in the system and manual intervention test categories. There might be some confusion when trying to remember which category of test 20 should be run to fit a particular need.

One recommended practice is to have each test number define one test or test sequence. The advantage is that the test number denotes a specific test that may be allowed to run in multiple categories. For example, test 20 might be an EEROM checksum test that is allowed to run in both the initialization and default extended test categories. If a request to run test 20 is given with the category field set to system test, the adapter would return UNIMPLEMENTED in the TEST_STATUS CSR. In any case, test 20 is always the EEROM checksum test.

Diagnostic firmware may use a test category table to determine which tests are run in specified categories. An X denotes that the test is allowed to run in the specified category.

The following table shows how a test may be allowed to run in multiple categories.

Table 5–4 Test Categories

	Initialize		Extended		Manual		System	
Test	**Init**	**Def Init**	**Ext**	**Def Ext**	**Man**	**Def Man**	**Sys**	**Def Sys**
1	X	X						
2	X							
3			X	X				
4			X					
5	X		X					

Continued on next page

It may be convenient to reserve a range of test numbers for each category using the test numbers shown in Table 5–5.

Table 5–5 Test Numbers

Test Decimal	Test Hexadecimal	Type of Tests
0	0x0000	Default tests in specified category
1-255	0x0001 - 0x00FF	Reserved
256-511	0x0100 - 0x01FF	Conformance tests
512-767	0x0200 - 0x02FF	Initialization tests
768-1023	0x0300 - 0x03FF	Extended tests
1024-1279	0x0400 - 0x04FF	Manual intervention tests
1280-1535	0x0500 - 0x05FF	System tests
1536-1791	0x0600 - 0x06FF	Scripts
65535	0xFFFF	All tests in specified category

Table 5–6 shows the result of combining the use of test categories and test numbers. Note that the tests are not limited to one category even though the test number may be in a specific range. For example, some initialization tests may also be run in the extended test category; however, the test number would still be in the range of initialization tests.

Continued on next page

Test Number Assignment Conventions, Continued

Table 5–6 Multiple Test Categories with Test Numbers

	Initialize		Extended		Manual		System	
Test	**Init**	**Def Init**	**Ext**	**Def Ext**	**Man**	**Def Man**	**Sys**	**Def Sys**
0x0100							X	
0x0200	X	X						
0x0201	X	X	X					
0x0202	X	X	X	X				
0x0203	X		X					
0x0204	X							
0x0300			X	X				
0x0301						X		

Chapter 6
Fault Management

Introduction

Chapter Overview

The arbitration and protocol chapters of the IEEE 896.1 standard specify behavior of nodes on the bus in detailed state equations. The state equations identify errors that may be detected during the different arbitration, connection, data, and disconnection phases.

The Profile B chapter of the IEEE 896.2 standard includes a subset of the arbitration and protocols specified in IEEE 896.1.

This chapter of the handbook describes the conventions for recording state upon detection of errors, recovery by retry, and reporting errors to the next level in a system.

The chapter also lists each error case separately and describes briefly the required practices for detecting errors, the capture of symptoms, and the signaling of the error to the software.

Fault Management Strategy

All nodes should capture the same error symptoms to support a workable fault management strategy. System bus bridges are expected to deviate in implementing some of the conventions for signaling detection of an error on the bus.

Futurebus+ Error Strategy

Three Aspects of Error Handling

The detection and handling of errors can be divided into three major categories.

- Parallel protocol

  Interpretation of the status wires

- Hardware-Level error symptom capturing

  Control and Status Registers (CSRs)

- Error reporting

  For example, interrupts

Errors

Errors that must be considered are:

- Connection phase

  Nonexistent address
  Address parity error
  Command parity error
  Tag parity error
  Protocol error [1]
  Transaction timeout
  Retry threshold exceeded
  Nonexistent transaction ID
  Unexpected split response[2]

Continued on next page

[1] Operation is defined and not restricted in IEEE 896.1, but is not supported by the module or profile. Unsupported command encodings may be reported as protocol errors.

[2] Not signaled on the bus. Reporting mechanism is unit or port architecture specific.

Errors (continued)

- Data phase

  Data parity error
  Tag parity error
  Command parity error
  Transaction timeout
  Protocol error

- Disconnection phase

  Data parity error
  Tag parity error
  Command parity error
  Transaction timeout
  Length error
  Protocol error

- Node internal errors and errors on the far side of a splitting bridge [3]
- Distributed arbitration and arbitrated message bus errors
- Split timeout

Profile B Errors

Conditions and status signals that are not supported in Profile B are:

- Wait status
- Intervention status
- Broadcast status

The assertion of these signals should be interpreted by Profile B masters as protocol errors.

Continued on next page

[3] Errors that may result in bad data on the bus.

Unsupported Profile B Errors

Errors flagged on Profile B unsupported options are:

- Transaction flag (conflict status)
- Tag parity error
- Distributed arbitration error

These errors may be ignored.

Non-Errors

Conditions and status signals that are not parallel protocol errors and are not addressed are:

- Busy status
- Power-Up reset
- Bus initialize
- System reset
- Live insertion
- Power-Related errors

Parallel Protocol Rules

Rules

This section explains the rules governing error strategy for Futurebus+ Profile B modules.

The general parallel protocol error signaling rules are as follows:

- Parallel protocol errors are signaled on the bus using Beat Error (BE*) or Transaction Error (TE*). Any module may signal an error in the connection phase. Only a node that is *participating* (or *selected*) may signal an error in the data or disconnection phases.
- A master or participating slave that detects an error or receives a parallel protocol signal reporting that another module on the bus detected an error should set the appropriate bits in the ERROR_HI and ERROR_LO error log registers. Note that only a slave is permitted to drive the status lines.
- Commands or operations that are unsupported by Profile B shall be treated as *protocol errors*.

Precise and Imprecise Errors

Precise Errors

Precise errors are signaled at the moment they occur. For example, a parity error on the D[ ]* lines causes an error that can be detected during the data beat containing the bad data.

Imprecise Errors

Imprecise errors are *not* signaled at the moment they occur; further progress may take place before the error is signaled. This may occur for either of the following reasons:

- The error signal is delayed because of timing considerations within the device detecting the error.
- The error is not related to a unique data beat, for example, split response timeout.

Master

Master Responsibilities

A master may detect an error in a connected or split transaction in any phase.

The master should save the command and address information it used in the connection phase and store it in the error log registers in the event that the transaction fails. The master should capture the status and capability lines at the time it detects a parallel protocol error signal (BE* or TE*), and should interrupt upper-level software.

In all cases, the master in a transaction should know about the error.[4] It is the responsibility of the master to notify the requester that an error was detected.

Connected Transactions

When an error is detected, or a parallel protocol error signal (BE* or TE*) is observed, the master should proceed to the disconnection phase.

The master is the only frame of reference that can detect a TRANSACTION_TIMEOUT error. In the event that the master detects this error, the master shall assert Reset (RE*) to cause a BUS_INITIALIZE. In this case, the master does not need to do anything extra to disconnect from the bus; asserting RE* will result in the initialization of the master state machine.

The master is the only frame of reference that can detect a RETRY_THRESHOLD_EXCEEDED error. (Retry threshold may be busy retry or error retry.) If the error is a BUSY_THRESHOLD_EXCEEDED error, the master shall disconnect from the bus and *not* drive any error status signal.

Continued on next page

[4] Exceptions: Slave-detected parity errors with parity error reporting disabled and back-bus (unit-specific) errors on a connected write transaction.

Split Responder in the Disconnection Phase

One special case is a master acting as split responder in the disconnection phase.

Errors detected on the far side of a splitting bridge or on a back bus of a module servicing a split request transaction are signaled by the split responder in the disconnection phase of the Read Response transaction. The specification for the status lines excludes using these lines as a path for signaling this type of error. The split transaction error state is signaled by the responder (master) on the AD[2..0]* lines during the disconnection phase of the Read Response transaction.

- A nonexistent memory error (NXM) on the far side of a splitting bridge is signaled by releasing AD2* (which represents the status of the SL* signal at the ultimate destination of the transaction). [5]
- A Conflict Status on the far side of a bridge is signaled by asserting AD1*.[6]
- Any other transaction error on the far side of a bridge is signaled by asserting AD0* (TE*).

The responder reporting the split transaction error should capture the disconnect data containing the error state in an ancillary error register. The master shall set the UNIT_SPECIFIC error bit in the ERROR_HI CSR.

[5] Not all bus bridge implementations send back a response in the case of an NXM. If the bus bridge does not send back a response, the result is a split timeout on the device waiting for the read response.

[6] Not all bus bridges implement the detection of this error.

Slave (Any Module)

Slave Responsibilities

The slave (any module) frame of reference is only relevant during the connection phase. Any module that detects a command or address parity error shall not assert SL* (selected) and shall drive BE*.

If no module drives SL* or BE* in the connection phase, the error has the same symptoms as a NONEXISTENT_MEMORY error (from the frame of reference of the master).

Participating Slave

Slave Responsibilities

A module that is *participating* should capture the command (CM[ ]*), status (ST[ ]*), and capability (CA[ ]*) lines in the ERROR_HI CSR at the time it signals an error. The module may also capture address (AD[ ]*) and data (D[ ]*) bus signals in ancillary error registers at the same time.

Connection Phase

A slave becomes *participating* when there are no parity errors on the address or the command and the address is one for which the module is the ultimate source and/or destination of the data. A selected slave shall drive SL* in the connection phase.

Connection phase errors are signaled by a selected module with BE*. While TE* is never used in the connection phase, it may still be asserted from the disconnection phase of the previous transaction. A participating slave shall assert BE* if there is a capability parity or protocol error.

Continued on next page

Data Phase

Precise errors may be signaled by a participating slave in the data phase with BE*. The conditions under which a participating slave shall assert BE* in the data phase are:

- Parity error reporting is enabled and a data or command parity error is detected on the current beat.
- A protocol error is detected.

Imprecise errors are signaled by a participating slave with TE*. The conditions under which a participating slave shall assert TE* in the data phase are:

- Parity error reporting is enabled and a data or command parity is detected; however, the report is delayed in time and was not signaled on the failing beat.
- TE* shall be asserted by a participating slave before the end of the disconnection phase if that participating slave asserted BE* at any time during the transaction.

NOTE
During the data phase, TE* can be asserted; however, a master is prohibited from sampling this status signal until the disconnection phase because of setup and hold timing limits.

- Participating slaves may use ED* to terminate the transaction in the case of an error, and signal the error using TE* in the disconnection phase.

Continued on next page

Participating Slave, Continued

Disconnection Phase

Errors are signaled by a participating slave in the disconnection phase with TE*. TE* is a catch-all status signal that allows a report on any error detected during the transaction no matter how late the error is detected. The conditions under which a participating slave shall assert TE* in the disconnection phase are:

- Parity error reporting is enabled and a disconnection data or command parity error is detected on the current beat.
- Parity error reporting is enabled and a data or command parity error is detected but the report is delayed in time and was not signaled in the data phase.
- Any error occurred internal to the participating slave that is not a bus error, but may have resulted in bad data on the bus.
- A protocol error is detected, for example, if the unrestricted data length is requested by the master (requester) in the disconnection phase of a split (request) transaction, or a packet mode length error was detected.

NOTE

If the data length is larger than the maximum a splitting slave is capable of supporting, but is not unlimited, no error is required to be signaled in the disconnection phase of the request transaction. In the response transaction the split responder should provide as much data as it is capable of providing (analogous to the use of ED*).

Summary

Among error indications, disconnection phase TE* is the lowest common denominator. It shall be asserted if any error is reported with BE* during the connection or data phases and if any error is detected but not reported with BE* in an earlier phase.

The participating slave may use BE* to report errors in the current data beat (yielding precise error reporting).

Hardware-Level Symptom Capturing

Errors Captured in CSRs

This section covers symptoms that are captured in CSRs and the conventions used for capturing and saving these symptoms.

- Error summary (ERROR_SUMMARY)

  This state indicates that symptoms of an error are captured in the ERROR_HI and ERROR_LO error log registers and possibly in the ancillary error registers.

- Transaction phase and frame of reference information

  Profile B required:
  MASTER

  Strongly recommended:
  CON_PHASE (connection phase)
  DATA_PHASE (data phase)
  DISCON_PHASE (disconnection phase)

- Error type (not always implicit in ST[7..0])

  Profile B required:
  PROTOCOL_ERROR
  COMMAND_PAR_ERROR
  DATA/ADDR_PAR_ERROR
  TRANSACTION_TIMEOUT
  SPLIT_TIMEOUT
  LENGTH_ERROR
  BUSY_RETRY_THRESHOLD_EXCEEDED

Continued on next page

Errors Captured in CSRs (continued)

NONEXISTENT_TRANSACTION_ID
unexpected split response
NONEXISTENT_ADDRESS

Strongly recommended:
UNIT_SPECIFIC_ERROR

- Bus signal lines recommended to be captured

  Profile B required:
  CAPABILITY_LINES (CA[2..0]*)
  STATUS_FIELD (ST[7..0]*)
  COMMAND_FIELD (CM[7..0]*, CP)

  Strongly recommended:
  FAILING_ADDRESS (AD[63..0]*)
  DATA_HOLD (D[255..64]*, AD[63..0]*)
  BYTE_LANE_IN_ERROR

- Other error state bits

  Strongly recommended:
  VENDOR_DEFINED (8)

Symptom Capturing

The bus master shall capture the state of the command signals and capability signals. It is recommended that the address also be captured during the connection phase. These signals shall be recaptured at the start of the next connection phase if no error is detected or reported using the status signals. If an error happens later on, the master freezes the previously captured state and any status lines at the time of error, and so on.

A participating slave shall capture error state and freeze the error log CSRs.

All other signals and the error state are captured by the master and the module that detected the error at the time the error is detected or reported.

Continued on next page

Split Response Disconnection Phase

Internal errors and errors on the far side of a splitting bridge are signaled during the disconnection phase using AD[2..0]*. These signals shall be checked by a participating slave during the disconnection phase of a split response.

Symptom Collection Software

It is important to note that the error symptom collection software for connection phase errors must collect symptoms and unlock error logs on all nodes on the local bus. Any node may detect and report an error during the connection phase and it is not possible to uniquely identify a single node as the only node driving BE*.

Futurebus+ Errors

Introduction

Error symptoms are dependent on the parallel protocol phase at the time the error is detected or signaled on the bus. Each phase is described individually; however, errors that can be detected in more than one phase are listed in each phase.

Connection Phase Errors

Nonexistent Address Errors

Error Description
This error is detected by the bus master when AI* is released by all slaves, and neither BE* nor SL* are asserted on the bus. This is a parallel protocol error and is observed by the master by decoding the status lines.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The error is detected by the absence of SL* asserted when AI* is released. The master immediately proceeds to the disconnection phase by releasing AS* (address strobe).

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
CON_PHASE	NONEXISTENT_ADD
CA[2..0]*	ST[7..0]*
CM[7..0]*	

In the master, the ERROR_LO CSR should have no error bits set.

In the master, the ancillary error registers should capture the target address.

Continued on next page

Nonexistent Address Errors (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Address Parity Error - Report Disabled

Error Description
This error is detected by any module decoding the address signals and the AW* command signal after AS* has been asserted. Parity is checked on the active byte lanes (as qualified by AW*).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
Slaves detecting bad address parity shall *not* assert SL*. Slaves with PARITY_ERROR_REPORTING_ENABLE clear shall *not* assert BE*.

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	DATA/ADDR_PARITY_ERROR_
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus and the byte lane in error.

Continued on next page

Address Parity Error - Report Disabled (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

If the module that detects the error was the target of the transaction, then the master will detect the same symptom as a NONEXISTENT_ADDRESS error. If a module that is not intended to participate in this transaction detects this, then no error is signaled on the status lines and the master will not be aware of the occurrence of the error. In either case, the symptom information will remain latched in the CSRs of the detecting slave until cleared by the software.

Continued on next page

Address Parity Error - Report Enabled

Error Description
This error is detected by any module decoding the address signals and the AW* command signal after AS* has been asserted. Parity is checked on the active byte lanes (as qualified by AW*).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The slave detecting bad parity shall *not* assert SL*. The slave shall signal the error by asserting BE* in the connection phase and by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	DATA/ADDR_PAR_ERROR
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus, and the byte lane in error.

Continued on next page

Address Parity Error - Report Enabled (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Command Parity Error - Report Disabled

Error Description
This error is detected by any module after AS* has been asserted. Parity is checked across all of the command lines (CM[7..0]*).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one clear).

Hardware-Level Error Signaling
Slaves detecting bad command parity shall *not* assert SL*. No slaves with PARITY_ERROR_REPORTING_ENABLE clear shall assert BE*.

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	COMMAND_PARITY_ERROR
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

If the module that detects the error was the target of the transaction, then the master will detect the same symptom as a NONEXISTENT_ADDRESS error. If a module that is not intended to participate in this transaction detects a COMMAND_PARITY_ERROR, then no error is signaled on the status lines and the master will not be aware of the occurrence of the error. In either case, the symptom information will remain latched in the CSRs of the detecting slave until cleared by the software.

Continued on next page

Command Parity Error - Report Enabled

Error Description
This error is detected by any module after AS* has been asserted. Parity is checked across all of the command lines (CM[7..0]*).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The slave detecting bad parity shall *not* assert SL*. The slave shall signal the error by asserting BE* in the connection phase and by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Continued on next page

Command Parity Error - Report Enabled (continued)

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	COMMAND_PARITY_ERROR
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Capability Error - Master Detected

Error Description
The master encounters this error when the slave asserts an unsupported capability on the CA[ ]* lines or an unsupported combination of status signals on the ST[ ]* lines.

These errors are a function of a specific profile or module characteristic. The following are examples of Profile B master capability errors:

- Split (CA2*-SR*) asserted during any transaction other than Read Unlocked or Read Partial (for example, Read Response, Write Partial, Write Unlocked)
- Wait (ST7*-WT*) asserted in the connection phase
- Intervention (ST5*-IV*) asserted in the connection phase
- Transaction Flag (ST4*-TF*) asserted in the connection phase
- Broadcast (ST3*-BC*) asserted in the connection phase

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The master shall proceed to the disconnection phase on detection of a master capability error. This error is *not* signaled on the bus on a status line (BE* and TE* are *not* asserted).

Continued on next page

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
CON_PHASE	PROTOCOL_ERROR
CA[2..0]*	ST[7..0]*
CM[7..0]*	

In the master, the ancillary error registers should capture the target address.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Capability Error - Slave Detected

Error Description
These errors are a function of a specific module characteristic, for example:

- A master asserts a command line that specifies an operation or data width not supported by a participating slave.
- A participating slave asserts a status line that specifies an operation not supported by another participating slave (IV*, BC*, TF*, SW*).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The participating slave shall assert BE* upon detection of a capability error. The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

Continued on next page

Capability Error - Slave Detected (continued)

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	CON_PHASE
PROTOCOL_ERROR	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Illegal Operation Error - Slave Detected

Error Description
This error is detected by a participating slave by decoding the command lines after AS* has been asserted.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

The following are examples of illegal operation errors:

- Reserved command encoding.
- The CM7* (AW*) command line is asserted in a Read Response command.
- The CM2* (SR*) capability line is asserted in a Read Response command.
- The DW is greater than 32 when the address target is CORE, ROM, or Futurebus+ reserved CSR space (except for a message passing area).
- Transaction other than Read Unlocked D32 or Write Unlocked D32 when the address target is CORE, ROM, or Futurebus+ reserved CSR space (except for a message passing area).

An Illegal Operation error is detected by any participating slave when the combination of address, command, and capabilities signals are decoded.

Continued on next page

Illegal Operation Error - Slave Detected (continued)

Hardware-Level Error Signaling
The participating slave that detected the error shall assert BE* in the connection phase. The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error symptom capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	CON_PHASE
PROTOCOL_ERROR	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Transaction Timeout - Master Detected

Error Description
Prior to or coincident with the start of a transaction, the master shall reset the transaction timer. A TRANSACTION_TIMEOUT error is detected when the transaction timer expires before the disconnection phase has ended. The duration of the TRANSACTION_TIMEOUT defaults to 128 μs $\pm$10%; however, it may be set to a different value by writing to the TRANSACTION_TIMEOUT CSR.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The master shall assert RE* to cause a BUS_INITIALIZE.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
CON_PHASE	TRANSACTION_TIMEOUT
CA[2..0]*	ST[7..0]*
CM[7..0]*	

In the master, the ancillary error registers should capture the target address.

Error Reporting
The master signals this event by initiating a BUS_INITIALIZE. In some systems this is likely to be a system fatal error; system bus bridges may turn the BUS_INITIALIZE into a SYSTEM_RESET.

Continued on next page

Busy Retry Exceeded

Error Description
The number of busy retry attempts has exceeded the value in the RETRY_THRESHOLD field of the BUSY_RETRY_COUNTER CSR.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
None. This error is not reported on a parallel protocol status signal.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the ERROR_SUMMARY and MASTER bits latched. The ERROR_LO CSR should freeze with the BUSY_RETRY_THRESHOLD_EXCEEDED bit latched.

In the master, the ancillary error registers should capture the target address.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Error Retry Exceeded

Error Description
The number of error retry attempts has exceeded the value in the RETRY_THRESHOLD field of the ERROR_RETRY_COUNTER CSR.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
None. This error is not reported on a parallel protocol status signal.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the ERROR_SUMMARY and MASTER bits latched. The ERROR_LO CSR should freeze with the ERROR_RETRY_THRESHOLD_EXCEEDED bit latched.

In the master, the ancillary error registers should capture the target address.

In the slave, the error CSRs may have detailed symptoms that indicate what error caused the last retry to fail.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Beat Error - Master Detected

Error Description
A master detects BE* asserted in the connection phase. This indicates that another module in the system detected a connection phase error and signaled the error event by asserting BE*.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The master, upon detection of BE*, proceeds to the disconnection phase by releasing AS*.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
CON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the error CSRs should have detailed symptoms that indicate why the Beat Error was asserted on the bus state set.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Unexpected Split Response or Nonexistent Transaction ID

Error Description
This error is either an unsolicited split response (transaction ID matches but there are no outstanding split transactions or a split timeout has occurred prior to the response), or a transaction ID mismatch error (there are outstanding split transactions but the transaction ID does not match any that are currently outstanding).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
None on the bus. This is not a Futurebus+ parallel protocol error. Notification that this error occurred is optional. A unit-specific mechanism should be used to signal this event.

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the ERROR_SUMMARY bit set. The ERROR_LO CSR should freeze with the NONEXISTENT_TRANSACTION_ID bit set.

In the slave, the ancillary error registers should have the failing address containing the transaction that is flagged nonexistent.

Error Reporting
The slave (requester) should signal this event by generating an error interrupt or by signaling the completion of the transaction with failing status.

Continued on next page

Wrong Split Response Data Width

Error Description
The responder in a split read transaction uses a different data width than specified in the command field of the request transaction.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The participating slave (original requester) that detected the error will assert BE* in the connection phase. The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Continued on next page

Wrong Split Response Data Width (continued)

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	PROTOCOL_ERROR
CA[2..1]*	ST[7..0]*
CM[7..0]*	

Error Reporting
The slave (requester) should signal this event by generating an error interrupt or by otherwise signaling the completion of this transaction with failing status.

NOTE

Conformance with the IEEE 896.2 standard requires inheritance of data width from request to response. Nodes are not required to check for this condition; however, if they do, this is the recommended error handling approach.

Data Phase Errors

Data Parity Error - Master Detected

Error Description
For read transactions, the master shall detect a DATA_PARITY_ERROR if a parity error is present during the data phase on the AD[ ]* or D[ ]* lines. Parity is checked on the active byte lanes (as qualified by DW[ ]*).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The master, upon detection of the error, should proceed to the disconnection phase by releasing AS*. The master and participating slave check parity on received data only.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DATA_PHASE	DATA/ADDR_PARITY_ERROR (a failed read transaction)
CA[2..0]*	ST[7..0]*
CM[7..0]*	

In the master, the ancillary error registers should capture the target address and the byte lane on which the error was detected.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Data Parity Error - Slave Detected - Report Disabled

Error Description
For write transactions, a participating slave shall detect a data parity error if a parity error is present during the data phase on the AD[ ]* or D[ ]* lines. Parity is checked on the active byte lanes (as qualified by DW[ ]*).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
A participating slave detecting bad parity with parity error reporting disabled shall not signal the error on the bus. The slave may assert ED* or may continue to consume bad data.

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	DATA/ADDR_PARITY_ERROR
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus and the byte lane on which the error was detected.

Error Reporting
The master may poll the slave to detect that an error has occurred. The slave may interrupt higher level software to communicate the error.

Continued on next page

Data Parity Error - Slave Detected - Report Enabled

Error Description
For write transactions, a participating slave shall detect a data parity error if a parity error is present during the data phase on the AD[ ]* or D[ ]* lines. Parity is checked on the active byte lanes (as qualified by DW[ ]*).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
A participating slave, upon detection of the error, may signal the event in the current data beat by asserting BE*. The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	DATA/ADDR_PARITY_ERROR
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus and the byte lane on which the error was detected.

Continued on next page

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Command Parity Error - Slave Detected - Report Disabled

Error Description
Modules shall detect DATA_COMMAND_PARITY_ERROR if a parity error is present during the data phase on the CM[ ]* lines. Even though the command is used only during the first data beat of a transaction, the parity on this field must be correct for every data beat.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
A participating slave detecting bad parity with parity error reporting disabled shall not signal the error on the bus.

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	DATA/ADDR_PARITY_ERROR
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus.

Error Reporting
The master may poll the slave to detect that an error has occurred. The slave may interrupt higher level software to communicate the error.

Continued on next page

Command Parity Error - Slave Detected - Report Enabled

Error Description
Modules shall detect DATA_COMMAND_PARITY_ERROR if a parity error is present during the data phase on the CM[ ]* lines. Even though the command is used only during the first data beat of a transaction, the parity on this field must be correct for every data beat.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
A participating slave, upon detection of the error, may signal the event in the current data beat by asserting BE*. The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	COMMAND_PARITY_ERROR
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus.

Continued on next page

Command Parity Error - Slave Detected - Report Enabled (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Transaction Timeout

Error Description
Prior to or coincident with the start of a transaction, the master shall reset the transaction timer. A TRANSACTION_TIMEOUT error is detected when the transaction timer expires before the end of the disconnection phase. The duration of TRANSACTION_TIMEOUT defaults to 128 μs $\pm$10%; however, it may be set to a different value by writing to the TRANSACTION_TIMEOUT CSR.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The master shall assert RE* to cause a BUS_INITIALIZE.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	
MASTER	TRANSACTION_TIMEOUT
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

Error Reporting
The master signals this event by initiating a BUS_INITIALIZE. In some systems this is likely to be a system fatal error; system bus bridges may turn the BUS_INITIALIZE into a SYSTEM_RESET.

Continued on next page

Illegal Operation or Capability Error - Slave Detected

Error Description
An example of this type of error is a slave write (SW*) CM4* asserted in a Profile B transaction.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The participating slave that detects this error may signal the event in the first data beat by asserting BE* and shall signal the event by asserting TE* until the end of the connection phase of the next bus transaction.

Locked Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	PROTOCOL_ERROR
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the data driven on the bus at the time the error was detected.

Continued on next page

Illegal Operation or Capability Error - Slave Detected (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Beat Error - Master Detected

Error Description
A master detects BE* asserted in the data phase. This indicates that another module in the system detected an error and signaled the error event by asserting BE*.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The master, upon detection of BE*, proceeds to the disconnection phase by releasing AS*.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DATA_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the error CSRs should have detailed error state indicating why the slave asserted BE*.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Disconnection Phase Errors

Data Parity Error - Slave Detected - Report Disabled

Error Description
Modules shall detect a DISCONNECTION_DATA_PARITY_ERROR if a parity error is present on the AD[ ]* lines during the disconnection phase when those lines are activated for specified information.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

No parity checking on disconnect data should be done on read transactions because the state on the data lines of the bus is undefined in the disconnection phase following a connected read transaction.

Hardware-Level Error Signaling
The participating slave detecting this error with parity error reporting disabled shall not signal the error on the bus.

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	ADDR/DATA_PAR_ERROR
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers may capture the target address of the transaction.

Error Reporting
The master may poll the slave to detect that an error has occurred. The slave may interrupt higher level software to communicate the error.

Continued on next page

Disconnection Phase Errors, Continued

Data Parity Error - Slave Detected - Report Enabled

Error Description
Modules shall detect a DISCONNECTION_DATA_PARITY_ERROR if a parity error is present on the AD[ ]* lines during the disconnection phase when those lines are activated for specified information.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

No parity checking on disconnect data should be done on read transactions because the state on the data lines of the bus is undefined in the disconnection phase following a connected read transaction.

Hardware-Level Error Signaling
The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

Continued on next page

Data Parity Error - Slave Detected - Report Enabled (continued)

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	ADDR/DATA_PAR_ERROR
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers may capture the target address of the transaction.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Command Parity Error - Slave Detected - Report Disabled

Error Description
Modules shall detect a DISCONNECTION_COMMAND_PARITY_ERROR if a parity error is present during the disconnection phase on the CM[ ]* lines.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The participating slave detecting a disconnection command parity error with parity error reporting disabled shall not signal the error on the bus.

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	COMMAND_PARITY_ERROR
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the disconnection data.

Error Reporting
The master may poll the slave to detect that an error has occurred. The slave may interrupt higher level software to communicate the error.

Continued on next page

Command Parity Error - Slave Detected - Report Enabled

Error Description
Modules shall detect a DISCONNECTION_COMMAND_PARITY_ERROR if a parity error is present during the disconnection phase on the CM[ ]* lines.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	COMMAND_PARITY_ERROR
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the disconnection data.

Continued on next page

Command Parity Error - Slave Detected - Report Enabled (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Transaction Timeout

Error Description
Prior to or coincident with the start of a transaction, the master shall reset the transaction timer. A TRANSACTION_TIMEOUT error is detected when the transaction timer expires before TRANSACTION_COMPLETE has set. The duration of TRANSACTION_TIMEOUT defaults to 128 μs $\pm$10%; however, it may default to a different value by writing to the TRANSACTION_TIMEOUT CSR.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
A master that detects a transaction timeout shall use the RE* line to initiate a BUS_INITIALIZE.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DISCON_PHASE	TRANSACTION_TIMEOUT
CA[2..0]*	ST[7..0]*
CM[7..0]*	

In the master, the ancillary error registers should capture the target address.

Error Reporting
The master signals this event by initiating a BUS_INITIALIZE. In some systems this is likely to be a system fatal error: system bus bridges may turn the BUS_INITIALIZE into a SYSTEM_RESET.

Continued on next page

Length Error - Master Detected - Packet Mode

Error Description
In packet mode, more or less data than expected is received.

Hardware-Level Error Signaling
The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DISCON_PHASE	LENGTH_ERROR
CA[2..0]*	ST[7..0]*
CM[7..0]*	

In the master, the ancillary error registers should capture the target address.

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Length Error - Slave Detected - Packet Mode

Error Description
In packet mode, more or less data than expected is received.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level error Signaling
The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	DISCON_PHASE
LENGTH_ERROR	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the disconnection data.

Continued on next page

Length Error - Slave Detected - Packet Mode (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Disconnection Phase Errors, Continued

Internal or Forwarded Error - Slave Detected

Error Description
When an internal error occurs on a slave and that error may result in data corruption, the error event shall be signaled on the bus.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
In a connected transaction, the slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the ERROR_HI CSR should freeze with the following bits latched.

ERROR_SUMMARY	UNIT_SPECIFIC_ERROR
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the slave, the ancillary error registers should capture the address driven on the bus.

Continued on next page

Internal or Forwarded Error - Slave Detected (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Internal or Forwarded Error - Split Responder Detected

Error Description
When an error occurs in a split transaction on the far side of a splitting bridge and an error or data corruption results, the error event shall be signaled on the bus. These errors are only detected and reported using this mechanism when the transaction has been split (into separate request and response actions). These errors may include *nonexistent address*, *conflict error* (end-to-end busy), and *transaction error* (any other error type).

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
This event is signaled with disconnection data transmitted by the master on the AD[2..0]* lines in the disconnection phase of a split response transaction.

Hardware-Level Error Symptom Capturing
In the master (the split responder), the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	MASTER
DISCON_PHASE	UNIT_SPECIFIC_ERROR (logical OR of the disconnect AD[2..0] lines)
CA[2..0]*	ST[7..0]*
CM[7..0]	

In the master, the ancillary error registers should capture the target address and the disconnection data that signaled the error event.

Continued on next page

Internal or Forwarded Error - Split Responder Detected (continued)

In the slave (the original requester), the ERROR_HI CSR may set no error state or it may have the following bits latched:

ERROR_SUMMARY	DISCON_PHASE
CA[2..0]*	ST[7..0]*
CM[7..0]*	

In the slave, the ancillary error registers should capture the disconnection data.

Error Reporting
The initial requester (in this case the slave) should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Disconnection Phase Errors, Continued

Transaction Error - Master Detected

Error Description
A master detects TE* asserted in the disconnection phase. This indicates that another module in the system detected an error and signaled the error event by asserting TE*.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The master, upon detection of TE*, should interrupt upper-level software.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched.

ERROR_SUMMARY	MASTER
DISCON_PHASE	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ancillary error registers should capture the target address.

In the slave, the error CSRs should have detailed error state indicating why it asserted TE*.

Continued on next page

Disconnection Phase Errors, Continued

Transaction Error - Master Detected (continued)

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Excess Response Data Error - Slave Detected

Error Description
The responder in a split read transaction returns more data than specified in the disconnection phase of the request transaction.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
None. The slave should use ED* to terminate the transaction.

Hardware-Level Error Symptom Capturing
Not enough symptoms can be captured to diagnose this error.

In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	PROTOCOL_ERROR
CA[2..1]*	ST[7..0]*
CM[7..0]*	

In the slave, the ERROR_LO CSR should have no state set.

In the slave, the ancillary error registers should capture the target address.

Error Reporting
The slave (original requester) should signal this event by generating an error interrupt or by signaling the completion of the transaction with failing status.

Continued on next page

Illegal Operation Error - Slave Detected

Error Description
This error occurs when the splitting slave receives unrestricted data length (DL = 0) in the disconnection data supplied by the master.

The bits that capture this state in the error log register remain set until cleared by a bus transaction (write-one to clear).

Hardware-Level Error Signaling
The slave shall signal the error by asserting and maintaining TE* until the end of the connection phase of the next bus transaction.

Hardware-Level Error Symptom Capturing
In the slave, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	DISCON_PHASE
PROTOCOL_ERROR	CA[2..0]*
ST[7..0]*	CM[7..0]*

In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY	DISCON_PHASE
CA[2..0]*	ST[7..0]*
CM[7..0]*	

In the master and slave, the ancillary error registers should capture the failing address and the disconnection data (DATA_ HOLD CSR).

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Other Errors

Split Timeout Error

Error Description
This error occurs when the elapsed time waiting for a split response exceeds the value stored in the SPLIT_TIMEOUT registers.

Hardware-Level Error Signaling
None. This error is not reported on a parallel protocol status signal.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze and have the following bits latched:

ERROR_SUMMARY	MASTER
SPLIT_TIMEOUT	

Error Reporting
The master should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

Continued on next page

Arbitration Bus Errors

Error Description
Errors detected on the arbitration bus are asynchronous with transactions on the parallel protocol bus. The two possible error types are:

- Arbitration timeout
- Arbitration comparison or parity error

Hardware-Level Error Signaling
Arbitration condition signals AC1* and AC0* are both asserted in phase 3 of an arbitration competition upon the detection of an error on the arbitrated message bus.

Hardware-Level Error Symptom Capturing
In the master, the ERROR_HI CSR should freeze with the following bits latched:

ERROR_SUMMARY MASTER

In the master, the ERROR_LO CSR should freeze with ARB_ERR and one of the following bits latched:

ARB_COMPARISON_OR_PARITY_ERR

ARB_TIMEOUT_ERROR

Error Reporting
The master-elect (or message sender) should signal this event to the software driver of this transaction by generating an error interrupt or by signaling completion of the transaction with failing status.

NOTE

Profile B modules are not required to connect to the arbitration bus. If Profile B modules use the arbitration bus for the detection of power fail, they are not required to detect errors on this bus.

Appendix A
CRC-16 Algorithm

CRC-16 Algorithm

Example

The CRC calculation will start initially with a 16-bit check value of zero and be updated by each 16-bit value protected by the CRC. The IEEE 1212 CSR Architecture standard provides the following example, written in the C language, to describe the CRC algorithm:

```
/* crc_16 calculation, performed on longwords at a time
crc_quadlet is initially called with check=0 and data= the
first longword to be checked;
 crc_quadlet returns newly calculated check value
for additional calculations;  */
short crc_quadlet(unsigned data, unsigned check)
{int sum;  /* integers are >=32 bits */
for (shift=28; shift>0; shift -=4)
 {  sum= ((check>>12)^(data>>shift))&0XF;
 check=(check<<4)^(sum(<<12)^(sum<<5)^(sum);
        }
return(check&0XFFFF);  /* the calculated CRC check value */
>}
```

Continued on next page

Appendix B
CSR Architecture ROM Directory Entries

Summary

CSR Architecture and ROM Directory Entries

Each entry in the ROM directory is identified as to its type by an 8-bit key value. This 8-bit key value is divided into the following two fields:

- The most significant 2 bits identify the key type.
- The least significant 6 bits identify a particular key definition.

The key types are:

- 0b00: Immediate value
- 0b01: Offset for an immediate value
- 0b10: Offset to a leaf
- 0b11: Offset to a directory

Table B–1 summarizes the ROM definition values defined in the IEEE 1212 CSR Architecture standard. The entries in the table are sorted by key definition values, that is, the least significant 6 bits.

Continued on next page

Table B–1 Key Definition Values

Value	Key Type	Root Directory Entry Descriptors
0x81	leaf OR	TEXTUAL_DESCRIPTOR
0xC1	directory	This key entry provides a pointer to either a leaf or a directory that describes (in human readable characters) a portion of a module, node, or unit. The TEXTUAL_ DESCRIPTOR entry occurs immediately after the directory entry that it describes.
0x82	leaf or	BUS_DEPENDENT_INFO
0xC2	directory	This key entry provides a pointer to either a leaf or a directory that contains information not found in the Bus Info Block
0x03	immediate	MODULE_VENDOR_ID This entry provides the company ID of the vendor manufacturing the module.
0x04	immediate	MOD_HW_VERSION This key can be used to identify the diagnostic software for extensive module tests or identification for IVP at installation verification.

Continued on next page

Table B–1 (Continued) Key Definition Values

0x05	immediate	MODULE_SPEC_ID This key provides the company ID of the vendor that defined the architectural interface for the module. If not implemented, the assumed value shall be equal to MODULE_VENDOR_ID.
0x06	immediate	MODULE_SW_VERSION This key specifies that the immediate value is used to identify the I/O driver for this module. Note that if this is a multi-node or multi-unit module with more than one I/O driver, this key will not be provided. A NODE_SW_VERSION (0x0B) in the Root directory, or UNIT_SW_KEY (0x13) in the Unit Directories will be provided in its place. All software version entries are mutually exclusive.

Continued on next page

Table B–1 (Continued) Key Definition Values

0x87 0xC7	leaf or directory	MODULE_DEPENDENT_ INFO This key specifies that the offset points to a leaf or directory with module vendor-dependent data.
0x08	immediate	NODE_VENDOR_ID This key specifies that the immediate value provides the 24-bit company ID of the vendor that manufactured the node. If not implemented, the assumed value shall be equal to that of MODULE_ VENDOR_ID.
0x09	immediate	NODE_HW_VERSION This key can be used to identify the diagnostic software for extensive node tests or identification for IVP at installation verification.

Continued on next page

Table B–1 (Continued) Key Definition Values

0x0A	immediate	NODE_SPEC_ID This key provides the company ID of the vendor that defined the architectural interface for the node. If not implemented, the assumed value shall be equal to that of NODE_VENDOR_ID.
0x0B	immediate	NODE_SW_VERSION This key specifies that the immediate value is used to identify the I/O driver for this node.
0x0C	immediate	NODE_CAPABILITIES This key specifies that the bits in the immediate entry indicate which options in the IEEE 1212 CSR Architecture standard are implemented in this node.

Continued on next page

Table B–1 (Continued) Key Definition Values

0x8D	leaf	NODE_UNIQUE_ID This entry points to a leaf containing a 64-bit binary number. If implemented, this number shall be uniquely assigned by the vendor to each node. (This could provide a serial number.)
0x0E 0x4E	immediate or CSR offset	NODE_UNITS_EXTENT This entry specifies (or points to) a CSR that specifies the size and alignment constraints for the extended units space of the node.

Continued on next page

Table B–1 (Continued) Key Definition Values

0x0F 0x4F	immediate or CSR offset	NODE_MEMORY_EXTENT This entry specifies (or points to) a CSR that specifies the size and alignment constraints for the extended memory space of the node.
0x90 0xD0	leaf or directory	NODE_DEPENDENT_INFO This key specifies that the offset points to a leaf or directory with node vendor-dependent data.
0xD1	directory	UNIT_DIRECTORY This key specifies an offset pointer to a Unit Directory. If implemented, there will be an entry pointing to a Unit Directory for each Unit. Note that if UNIT_ DIRECTORY keys are implemented, then neither a MODULE_SW_VERSION or NODE_SW_VERSION entry can be present according to the IEEE 1212 CSR Architecture standard, ROM section.

Continued on next page

Table B–1 (Continued) Key Definition Values

0x12	immediate	UNIT_SPEC_ID This key provides the company ID of the vendor that defined the architectural interface for the unit. If not implemented, the assumed value shall be equal to NODE_VENDOR_ID.
0x13	immediate	UNIT_SW_VERSION This key specifies that the immediate value is used to identify the I/O driver for this unit.
0x94 0xD4	leaf or directory	UNIT_DEPENDENT_INFO This key specifies that the offset points to a leaf or directory with UNIT_ VENDOR_DEPENDENT.
0x95	leaf	UNIT_LOCATION This entry provides (base and boundary) address mapping information about the unit into the node address space.

Continued on next page

Summary, Continued

Table B–1 (Continued) Key Definition Values

0x16	immediate	UNIT_POLL_MASK This entry specifies a 16-bit poll mask value to define bits in the STATE_CLEAR CSR that are affected by this unit.

Glossary of Terms

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

A32 Transaction: A transaction where 32 bits of addressing information are provided during the connection phase.

A64 Transaction: A transaction where 64 bits of addressing information are provided during the connection phase.

Arbitration: The process of selecting the next bus master.

Arbitration Message: An event number broadcast on the arbitration bus to all modules on the arbitration bus.

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

Big Addressan: A method of presenting addresses on a multiplexed address and data bus such that the most significant address bits travel on the same lines as the data accessed at the lowest address.

Big Endian: A method of storing multibyte data in a byte-addressable memory such that the most significant byte of the data is stored at the lowest address. The VAX architecture stores floating-point numbers in big endian format.

BIST: Built-In-Self-Test—the ability of a device to test itself and determine whether or not it is functioning.

Board: See *Printed Wiring Board.*

BTL: Backplane transceiver logic—the family of semiconductor logic devices used as the transceivers on Futurebus+ Application Environment Profiles A, B, and F.

Bus Bridge: A Profile B module that interfaces between Futurebus+ and another bus or data channel. See *System Bus Bridge and I/O Bus Bridge*.

Bus Master: The device that is currently permitted to control the parallel protocol portion of the Futurebus+.

Bus Terminator: See *Terminator.*

Bus Tenure: The duration that the master controls the bus; that is, the total time that a module has the right to initiate bus transactions.

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

Busy: If a slave cannot or should not accept a bus transaction, it, or any other module, may issue a Busy status to the master of the transaction. The master must then enter the disconnection phase. The master may retry the transaction after a suitable time interval. During this delay, the master usually releases the bus to any pending master-elect.

Byte: A set of eight adjacent binary digits (bits).

Byte Lane: A data path formed by eight data lines and one parity line used to carry a single byte among the system modules.

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

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

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

Coherence Line: A data block for which cache consistency attributes are maintained.

Compelled Data Transfer Protocol: A technology-independent transfer mechanism is referred to as compelled because the slave is compelled to provide a response before the master proceeds to the next transfer.

Competitor: A module actively participating in the arbitration process.

Configuration Process: The process selected to manage the configuration and initialization of all modules on that logical bus, or the process selected to direct the configuration and initialization of an entire system with multiple interconnected logical buses.

Configuration process is the terminology used in Digital systems to represent the system bus bridge. Configuration process, as well as diagnostic process, are used in this handbook in place of the IEEE term *Monarch*.

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

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

Control Acquisition: The total of all bus activity associated with acquiring exclusive control of the bus.

Coverage: See *Diagnostic Coverage.*

CSA: Canadian Standards Association—a regulatory agency responsible for establishing safety standards in Canada.

CSR: Control and Status Register.

CSRA: Control and Status Register Architecture. (See the IEEE 1212 standard.)

Data Phase: The period within a transaction used to transfer data.

D32 Transaction: A transaction where 32 bits (4 bytes) of data are moved in each data beat.

D64 Transaction: A transaction where 64 bits (8 bytes) of data are moved in each data beat.

D128 Transaction: A transaction where 128 bits (16 bytes) of data are moved in each data beat.

D256 Transaction: A transaction where 256 bits (32 bytes) of data are moved in each data beat. This is not supported on Profile B.

Deadlock: Deadlock occurs when actors are waiting on actions that can only be performed by those already waiting.

Delta_Vgnd: The voltage differential on 0 Vdc rail between any two worst case points of use on the backplane, including terminating resistors and backplane capacitors.

Diagnostic Coverage: The percentage of faults that can be detected by some specified diagnostic means, for example, built-in self-test.

Diagnostic Process: The process selected to manage the configuration and initialization of all modules on that logical bus, or the process selected to direct the configuration and initialization of an entire system with multiple interconnected logical buses.

Diagnostic process is the terminology used in Digital systems to represent the system bus bridge. Diagnostic process, as well as configuration process, are used in this handbook in place of the IEEE term *Monarch*.

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

Double Hexaword: A data quantity of 64 adjacent bytes. This is equivalent to 16 data beats of a D32 transaction or 8 data beats of a D64 transaction.

Forward Progress: A module that is not blocked from performing the tasks necessary to achieve its goal is said to be making forward progress. Forward progress is guaranteed only in the absence of deadlock or starvation.

Geographical Address: A unique identifier statically assigned to each logical module slot on the bus and assumed by any module connected to that slot.

Glitch: A momentary, usually undesirable pulse in a signal.

Glitch Filter: A means of eliminating or ignoring glitches.

Hexaword: A data quantity of 32 adjacent bytes. This is equivalent to 8 data beats of a D32 transaction or 4 data beats of a D64 transaction.

IEC: International Electrotechnic Committee—a body responsible for establishing international electronic standards.

IEEE: The Institute of Electrical and Electronic Engineers, Inc.—the body responsible for establishing the IEEE 896 family of Futurebus+ standards.

Imprecise Error: An error signaled at some time after its original occurrence, such that the error cannot be attributed to a single, unique instruction or bus cycle.

Incident-Wave Switching: An electrical signaling method that is able to correctly detect the incidence (the first wavefront) of an electrical signal propagating along a bus line. Contrast this with methods that do not detect the signal until one or more reflections have propagated back and forth along the bus line.

Institute of Electrical and Electronic Engineers, Inc.: See *IEEE.*

International Electrotechnic Committee: See *IEC.*

I/O Bus Bridge: In the context of this handbook, any device that connects (bridges) from the Futurebus+ to another I/O bus such as the VMEbus or Q–bus.

Intervening Slave: The participating slave that, although not the repository of last resort of the requested data, finds it necessary to prevent the repository of last resort from providing the requested data. Having done so, the intervening slave provides the data instead. This is not supported in Profile B.

Little Addressan: A method of presenting addresses on a multiplexed address and data bus such that the least significant address bits travel on the same lines as the data accessed at the lowest address. The Futurebus+ is little addressan.

Little Endian: A method of storing multibyte data in a byte-addressable memory such that the least significant byte of the data is stored at the lowest address. The VAX architecture stores integers in little endian format.

Livelock: Livelock is a metastable situation in which some modules acquire and release resources in a way that none of them make progress.

Locking: A facility whereby a module is requested to guarantee exclusive access to addressed data, blocking other modules from accessing that data. This allows indivisible operations to be performed on addressed resources. This is not supported in Profile B.

Longword: A data quantity of 4 adjacent bytes. This is equivalent to a single data D32 transaction. A D64 transaction would require a Read Partial or Write Partial transaction.

Lstub: The stub length, measured from the connector PTH to the center of the transceiver pin surface-mount pad (include etch length from dispersion PTH, if present).

Master: A module that acquires control of the bus by receiving a grant from the arbiter and proceeding with a bus transaction.

Master Elect: A module that has received a grant and is now waiting for the current bus master to release the bus.

Megabyte: 2^20 (1,048,576) bytes of data.

Mixed Endian: The storage of data in both big endian and little endian formats. For example, in the VAX architecture, floating point numbers are stored in big endian format while integers are stored in little endian format.

Module: A board, or board set, consisting of zero, one or two nodes, that share a physical interface with the backplane. Although only one board in a module connects to bus signals, each board connector could distribute board-local power.

Monarch: The term used by the IEEE specifications to refer to the software that is responsible for initializing and configuring (governing) Futurebus+ systems.

In Digital systems, this term is not used. Instead the term initialization and configuration process is used.

Natural Alignment: Naturally aligned data is defined to be a data quantity of 2n bytes whose least significant n address bits are zero. Thus, a naturally aligned quadword (8 adjacent bytes) of data would have the lower 3 address bits equal to zero.

Node: A logical entity on the Futurebus+ bus that has a reserved address space for CSRs.

Octaword: A data quantity of 16 adjacent bytes. This is equivalent to 4 data beats of a D32 transaction or 2 data beats of a D64 transaction.

Octet, also Octlet: A set of 8 adjacent bytes.

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

Participating Slave: A slave involved in a transaction as either a selected slave, an intervening slave, a broadcast slave, or a slave involved in multiple packet mode.

Port Architecture: See *Unit Architecture.*

Potential Master: A module capable of acquiring control of the bus through the control acquisition process.

Precise Error: An error that can be attributed to a single, unique instruction or bus cycle.

Preemption: Preemption occurs when the current master is told to end its transaction in favor of the master-elect.

Printed Wiring Board (PWB): That physical board that supports the electronic and mechanical components of an electronic circuit. The electronic components are then connected by wiring traces printed upon (and within) the board.

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

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

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

Quadlet: Four bytes (32 bits).

Quadword: A data quantity of 8 adjacent bytes. This is equivalent to 2 data beats of a D32 transaction or 1 data beat of a D64 transaction.

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

In the context of central arbitration, request also means to ask for bus tenure.

Requester: The module that initiated a transaction by sending a request (containing address, command, and sometimes data).

Responder: The module that completes a transaction by sending a response (containing the completion status and sometimes data).

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

Rsig: The resistance of the signal etch.

Rterm: The resistance of the terminator.

Stub: The very short signal path on the module connecting the BTL transceiver to the Futurebus+ connector.

Selected Slave: A slave selected by the master, by virtue of its recognition of the address on the bus lines, during the connection phase.

Shared Memory: The address space in the system accessible to all caching modules.

Skew: The difference between the propagation delays of two or more signals causing signals sent simultaneously to arrive at the destination nonsimultaneously ("skewed" in time).

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

Split Transaction: An operation in which the request is transmitted in one bus transaction and the response is transmitted in a separate, subsequent bus transaction.

Starvation: A system condition that occurs when one or more modules perform no useful work for an indefinite period of time due to lack of access to the bus or other system resources.

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

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

Terabyte: 2^40 (1,099,511,627,776) bytes of data

Terminator: Those devices used at one or both ends of an electrical line to absorb the incident electrical wave and prevent reflection of that wave. The terminator is typically a pure resister that has a value equal to the characteristic impedance (Z_0) of the transmission line.

Transfer Line Size: The size of the block of data transferred to or from main memory in a caching environment.

UL: Underwriters Laboratories—a regulatory organization that sets safety standards for equipment sold in the United States.

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

Unit Architecture: The specification document describing the format and function of the unit's software-visible CSRs.

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

VDE: A regulatory organization that sets safety and other standards for equipment sold in Germany.

Vterm: The terminator source voltage to be maintained under all load conditions. Vterm is measured from any connector ground pin to the voltage pin on any terminating resistor.

Wire-OR Glitch: A particular type of glitch that occurs on a wire-OR'd bus line when several drivers have been asserting the line and one of those drivers subsequently releases the line.

Z_0: The characteristic impedance of signal lines, including connector vias, on the unpopulated backplane.

$Z_{0'}$: The characteristic impedance of signal lines on a backplane populated by male connectors, with termination resistors and no modules inserted.

Zstub: The characteristic impedance of the signal line stub on the module.

Index

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z