EISA System Architecture

Second Edition

MINDSHARE, INC.
TOM SHANLEY
DON ANDERSON

Addison-Wesley Publishing Company
Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam
Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan
Paris • Seoul • Milan • Mexico City • Taipei

MindShare, Inc (r)
SINGLE-USER LICENSE AGREEMENT

Please read this document carefully before proceeding. This Agreement licenses this electronic book to you and contains warranty and liability disclaimers. By viewing this book, you are confirming your acceptance of the book and agreeing to become bound by the terms of this Agreement. If you do not wish to do so, immediately return the book to MindShare, Inc.

1. DEFINITIONS

(a) "book or electronic book" means the electronic book covered by this Agreement, and any related updates supplied by MindShare, Inc. The book consists of the encrypted PDF file supplied in electronic form.

2. LICENSE
This Agreement allows the **SINGLE END-USER** to:

(a) View the book on a computer or a stand-alone ebook viewer.
(b) You may make and distribute copies of the book and electronically transfer the book from one computer to another or over a network.
(c) Certain rights are not granted under this Agreement, but may be available under a separate agreement. If you would like to enter into a Site or Network License, please contact MindShare.

3. RESTRICTIONS
(a) You may not copy screen images of the book, or any portion thereof.
(b) You may not decompile, reverse engineer, disassemble, or otherwise reduce the book to a human-perceivable form.
(c) You may not modify, rent, resell for profit, distribute or create derivative works based upon the book or any part thereof.
(d) You will not export or reexport, directly or indirectly, the book into any country prohibited by the United States Export Administration Act and the regulations thereunder.
(e) The book may not be used in a group viewing environment.

4. OWNERSHIP

The foregoing license gives you limited rights to use the book. You do not become the owner of, and MindShare retains title to, the intellectual property contained within the book, and all copies thereof. All rights not specifically granted in this Agreement, including Federal and International Copyrights, are reserved by MindShare.

5. DISCLAIMER OF WARRANTIES AND OF TECHNICAL SUPPORT:
The book is provided to you on an "AS IS" basis, without any technical support or warranty of any kind from MindShare including, without limitation, a warranty of merchantability, fitness for a particular purpose and non-infringement. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. YOU MAY ALSO HAVE OTHER LEGAL RIGHTS WHICH VARY FROM STATE TO

STATE. These limitations or exclusions of warranties and liability do not affect or prejudice the statutory rights of a consumer; i.e., a person acquiring goods otherwise than in the course of a business.

6. LIMITATION OF DAMAGES:
MINDSHARE SHALL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES OR LOSS (INCLUDING DAMAGES FOR LOSS OF BUSINESS, LOSS OF PROFITS, OR THE LIKE), WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, EVEN IF MINDSHARE OR ITS REPRESENTATIVES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. The limited warranty, exclusive remedies and limited liability set forth above are fundamental elements of the basis of the bargain between Mindshare and you. You agree that Mindshare would not be able to provide the book on an economic basis without such limitations.

7. GOVERNMENT END USERS (USA only):
RESTRICTED RIGHTS LEGEND The book is "Restricted Computer Software." Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in this Agreement and as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013 (OCT 1988), FAR 12.212(a)(1995), FAR 52.227-19, or FAR 52.227-14, as applicable." Manufacturer: Mindshare, Inc., 4285 Slash Pine Drive, Colorado Springs, CO 80908.

8. GENERAL:
This Agreement shall be governed by the internal laws of the State of Colorado. This Agreement contains the complete agreement between the parties with respect to the subject matter hereof, and supersedes all prior or contemporaneous agreements or understandings, whether oral or written. All questions concerning this Agreement shall be directed to: Mindshare, Inc., 4285 Slash Pine Drive, Colorado Springs, CO 80908, Attention: Chief Financial Officer.

Mindshare is registered trademark of Mindshare, Inc.

Single-User License Agreement 9/8/00.

Library of Congress Cataloging-in-Publication Data

Sponsoring Editor: Keith Wollman
Project Manager: Eleanor McCarthy
Production Coordinator: Lora L. Ryan
Cover design: Barbara T. Atkinson
Set in 10 point Palatino by MindShare, Inc.

1 2 3 4 5 6 7 8 9 -MA- 9998979695
First printing, February 1995

Addison-Wesley books are available for bulk purchases by corporations, institutions, and other organizations. For more information please contact the Corporate, Government, and Special Sales Department at (800) 238-9682.

This book is dedicated to my son, Ryan, a ray of sunshine who possesses an incredible amount of that rarest of elements, common sense.

Contents

Chapter 6: ISA Bus Cycles

Chapter 7: EISA CPU and Bus Master Bus Cycles

Chapter 8: EISA DMA

Chapter 9: EISA System Configuration

Part Two – Intel 82350DT EISA Chipset

Chapter 10: EISA System Buses

Chapter 11: Bridge, Translator, Pathfinder, Toolbox

Chapter 12: Intel 82350DT EISA Chipset

Figures

Tables

Acknowledgments

This book would not have been possible without the input of thousands of hardware and software people at companies such as Intel, Compaq, IBM and Dell over the past seven years. They constantly sanity-check me and make me tell the truth.

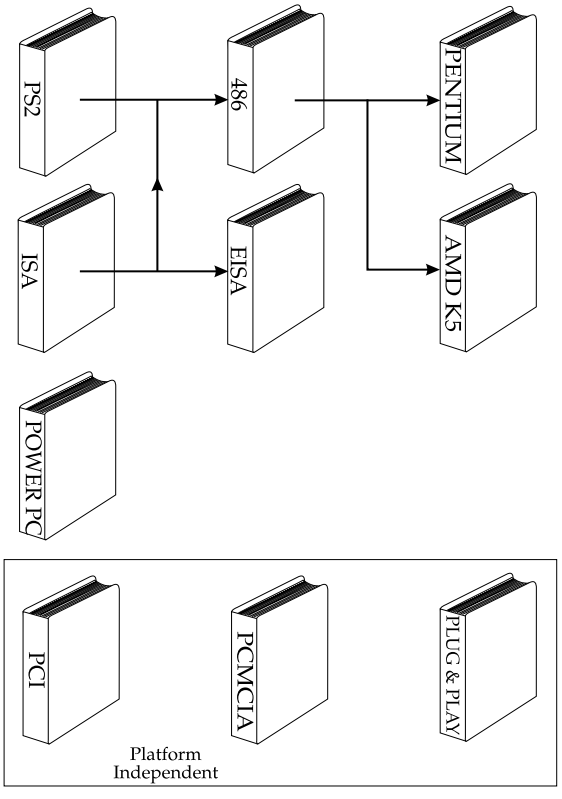
Special Thanks

Special thanks to Don Anderson for his constant help, advice and friendship

The MindShare Architecture Series

The MindShare Architecture book series includes: *ISA System Architecture*, *EISA System Architecture, 80486 System Architecture, PCI System Architecture*, *Pentium System Architecture, PCMCIA System Architecture, PowerPC System Architecture, Plug-and-Play System Architecture*, and *AMD K5 System Architecture.*

Rather than duplicating common information in each book, the series uses the building-block approach. *ISA System Architecture* is the core book upon which the others build. The figure below illustrates the relationship of the books to each other.



Series Organization

Organization of This Book

EISA System Architecture is divided into two major parts:

- Part One — The EISA Specification
- Part Two — The Intel 82350DT EISA Chip Set

Part One provides a detailed explanation of the ISA enhancements as set forth in the EISA specification, while Part Two provides a detailed description of the features implemented by the Intel 82350DT chip set. The following paragraphs provide a summary of each section.

Part One – The EISA Specification

EISA Overview

This chapter provides an overview of the benefits provided by the EISA extension to ISA.

EISA Bus Structure Overview

This chapter introduces the EISA bus structure and its relationship to the system board and expansion cards. The concepts of master and slave are introduced and defined. The types of bus masters and slaves are identified.

EISA Bus Arbitration

The bus arbitration scheme used by the EISA Central Arbitration Control is described in detail.

Interrupt Handling

An in-depth discussion of interrupt request handling in the ISA environment can be found in the chapter entitled "Interrupt Handling" in the MindShare book entitled *ISA System Architecture*. This chapter provides a brief review of the ISA interrupt request handling method and a detailed description of the EISA method.

Detailed Description of EISA Bus

This chapter provides a description of all the signals on the EISA bus.

ISA Bus Cycles

This chapter provides a review of the ISA bus master and DMA bus cycles.

EISA CPU and Bus Master Bus Cycles

This chapter provides a detailed description of the EISA CPU and bus master bus cycle types.

EISA DMA

This chapter describes the EISA DMA capability. This includes a description of the EISA DMA bus cycle types and the other improved capabilities of the EISA DMA controller.

EISA System Configuration

In this chapter, EISA automatic system configuration is discussed. This includes a description of the slot-specific I/O address space, the EISA product identifier, and the EISA card control ports. The EISA configuration process and board description files are also covered.

Part Two – The Intel 82350DT EISA Chipset

EISA System Buses

This chapter describes the major buses found in virtually all EISA systems. This includes the host, EISA, ISA and X buses.

Bridge, Translator, Pathfinder, Toolbox

This chapter provides a description of the major functions performed by the typical EISA chip set. It acts as the bridge between the host and EISA buses. It translates addresses and other bus cycle information into the form understood by all of the host, EISA and ISA devices in a system. When necessary, it performs data bus steering to ensure data travels over the correct paths between

the current bus master and the currently addressed device. It incorporates a toolbox including all of the standard support logic necessary in any EISA machine. It should be noted that the ISA bus is a subset of the EISA bus.

Intel 82350DT EISA Chip Set

This chapter provides an introduction to the Intel 82350DT EISA chip set. The focus is on the 82358DT EISA Bus Controller (EBC) the 82357 Integrated Systems Peripheral (ISP) and the 82352 EISA Bus Buffers (EBBs).

Who This Book Is For

This book is intended for use by hardware and software design and support personnel. Due to the clear, concise explanatory methods used to describe each subject, personnel outside of the design field may also find the text useful.

Those interested only in the compatibility and performance-related issues can skip over the detailed discussions and home in on the issues that interest them. Those interested in a more detailed explanation of the logic behind the enhancements can read the detailed explanations of bus cycle types and the EISA chip set.

Prerequisite Knowledge

EISA stands for the **E**xtension to the **I**ndustry **S**tandard **A**rchitecture. In order to fully grasp the EISA extensions, it is necessary to first understand the ISA system architecture. The detailed description of EISA presented in this book builds upon the concepts introduced in MindShare's book entitled *ISA System Architecture* to provide a clear, concise explanation of the EISA environment.

Documentation Conventions

This section defines the typographical conventions used throughout this book.

Hex Notation

All hex numbers are followed by an "h." Examples:

9A4Eh
0100h

Binary Notation

All binary numbers are followed by a "b." Examples:

0001 0101b
01b

Decimal Notation

When required for clarity, decimal numbers are followed by a "d." Examples:

256d
128d

Signal Name Representation

Each signal that assumes the logic low state when asserted is followed by a pound sign (#). As an example, the REFRESH# signal is asserted low when the refresh logic runs a refresh bus cycle.

Signals that are not followed by a pound sign are asserted when they assume the logic high state. As an example, DREQ3 is asserted high to indicate that the device using DMA Channel three is ready for data to be transfered.

Bit Field Identification (logical bit or signal groups)

All bit fields are designated as follows:

[X:Y],

where X is the most-significant bit and Y is the least-significant bit of the field. As an example, the ISA data bus consists of SD[15:0], where SD0 is the least-significant and SD15 the most-significant bit of the field.

We Want Your Feedback

MindShare values your comments and suggestions. You can contact us via mail, phone, fax or internet email.

Phone (800) 633-1440
Fax (719) 487-1434
Email tom@mindshare.com

Web Site

Because we are constantly on the road teaching, we can be difficult to get hold of. To help alleviate problems associated with our migratory habits, we have a web site to supply the following services:

- Download course abstracts.
- Download tables of contents of each book in the series.
- Facility to inquire about public architecture seminars.
- Message area to log technical questions.
- Message area to log suggestions for book improvements.
- Facility to view book errata and clarifications.

Web Site: www.mindshare.com

Mailing Address

MindShare, Inc.
4285 Slash Pine Dr.
Colorado Springs, CO 80908

PART ONE

THE EISA SPECIFICATION

Chapter 1

This Chapter

This chapter provides an overview of the benefits provided by the extension to ISA, EISA.

The Next Chapter

The next chapter, "EISA Bus Structure Overview," provides an overview of the "equal-opportunity" environment existent within all EISA-based systems. The different types of bus masters and slaves are identified.

Introduction

EISA is a superset of the ISA 8 and 16-bit architecture, extending the capabilities of ISA while still maintaining compatibility with ISA expansion boards.

EISA introduces the following improvements over ISA:

- Supports intelligent bus master expansion cards.
- Improved bus arbitration and transfer rates.
- Facilitates 8, 16 or 32-bit data transfers by the main CPU, DMA and bus master devices.
- An efficient synchronous data transfer mechanism, permitting single transfers as well as high-speed burst transfers.
- Allows 32-bit memory addressing for the main CPU, Direct Memory Access (DMA) devices and bus master cards.
- Shareable and/or ISA-compatible handling of interrupt requests.
- Automatic steering of data during bus cycles between EISA and ISA masters and slaves.
- 33MB/second data transfer rate for bus masters and DMA devices.
- Automatic configuration of the system board and EISA expansion cards.

Compatibility With ISA

EISA systems maintain full backward compatibility with the ISA standard. EISA connectors are a superset of the 16-bit connectors on ISA system boards, permitting 8 and 16-bit ISA expansion cards to be installed in EISA slots. While maintaining full compatibility with ISA expansion boards and software, EISA also offers enhancements in performance and functionality for EISA boards as well as some ISA boards.

Memory Capacity

EISA systems support a 32-bit address bus. The main CPU, bus master expansion cards and DMA devices may access the entire 4GB memory space. ISA memory expansion cards can be used without modification to populate the lower sixteen megabytes. EISA memory expansion cards can add as much memory as needed for the application, up to the theoretical maximum of 4GB.

Synchronous Data Transfer Protocol

The EISA bus uses a synchronous transfer protocol. Bus master cards, DMA and the main processor synchronize their bus cycles to the bus clock. The synchronous transfer protocol also provides the cycle control necessary to execute burst cycles with a transfer rate of up to 33 MB/second.

EISA provides a number of bus cycle types covering a range of transfer speeds for different applications. The standard bus cycle requires two bus clock cycles, while the main CPU, DMA and bus masters are permitted to generate burst cycles requiring one clock cycle per transfer.

Enhanced DMA Functions

EISA systems provide a number of DMA enhancements, including the ability to generate 32-bit addresses, 8, 16, and 32-bit data transfers and more efficient arbitration and data transfer types. In addition to newer, more efficient transfer types, EISA DMA also provides ISA-compatible modes with ISA timing and function as the default.

DMA offers a low-cost alternative to intelligent bus master cards. The EISA DMA functions are intended for I/O devices that do not require local intelligence on the I/O expansion card.

EISA 32-bit address support enables ISA and EISA DMA devices to transfer data to or from any 32-bit memory address. The default ISA DMA mode supports ISA-compatible 24-bit address generation with no software or hardware modifications. DMA software can be modified to support the 32-bit memory space, without modifications to the DMA hardware.

Any DMA channel may be programmed to perform 8, 16 or 32-bit data transfers. An 8-bit DMA device uses the lower data bus path, SD[7:0], while a 16-bit device uses the lower two paths, SD[7:0] and SD[15:8]. 32-bit DMA devices use all four data paths.

Using burst bus cycles, a 32-bit DMA device can transfer data at speeds up to 33 MB/second.

EISA DMA channels may be programmed to use one of four DMA bus cycle types when transferring data between the I/O device and memory. The default DMA bus cycle type, ISA-compatible, delivers a higher data transfer rate than ISA-compatible computers. The improvement is the result of EISA's faster bus arbitration and requires no hardware or software modifications to ISA-compatible DMA devices. Type A and B cycles are EISA modes that permit some ISA-compatible DMA devices to achieve higher performance. The burst DMA (Type C) bus cycle type is the highest-performance DMA bus cycle and is only available to DMA devices designed specifically for EISA burst.

Bus Master Capabilities

EISA-based systems support intelligent EISA bus master cards, providing data rates up to 33 megabytes/second using EISA burst bus cycles. A bus master card typically includes an on-board processor and local memory. It can relieve the burden on the main processor by performing sophisticated memory access functions, such as scatter/gather block data transfers. Examples of applications that might benefit from a bus master implementation include communications gateways, disk controllers, LAN interfaces, data acquisition systems and certain classes of graphics controllers.

Data Bus Steering

The EISA bus system permits EISA and ISA expansion cards to communicate with each other. Special system board logic ensures that data travels over the appropriate data paths and translates the control signals when necessary.

The system board data bus steering logic provides the automatic steering and control signal translation for ISA to ISA, EISA to EISA, ISA to EISA and EISA to ISA transfers.

Bus Arbitration

The EISA system board logic also provides a centralized arbitration scheme, allowing efficient bus sharing among the main CPU, multiple EISA bus master cards and DMA channels. The centralized arbitration supports preemption of an active bus master or DMA device and can reset a device that does not release the bus after preemption.

The EISA arbitration method grants the bus to DMA devices, DRAM refresh, bus masters and the main CPU on a fair, rotational basis. The rotational scheme provides a short latency for DMA devices to assure compatibility with ISA DMA devices. Bus masters and the CPU, which typically have buffering available, have longer, but predictable latencies.

Edge and Level-Sensitive Interrupt Requests

In order to provide backward-compatibility with ISA systems, EISA systems support positive edge-triggered interrupts. Unlike ISA systems, however, any EISA interrupt channel can be individually programmed to recognize either shareable, level-triggered or non-shareable, positive edge-triggered interrupt requests. Level-triggered operation facilitates the sharing of a single system interrupt request line by a number of I/O devices. Level-triggered interrupts might be used, for example, to share a single interrupt request line between a number of serial ports.

Automatic System Configuration

EISA systems implement the capability to perform automatic configuration of system resources and EISA expansion boards each time the system is powered

up. System resources such as serial ports, parallel ports, VGA and other manufacturer-specific functions can be fully configured programmatically.

The EISA expansion card manufacturer includes a configuration file with each expansion card shipped. The configuration files can be included with either new, fully-programmable EISA boards or switch-configured ISA or EISA products. The configuration files are used at system configuration time to automatically assign global system resources (such as DMA channels and interrupt levels), thus preventing resource conflicts between the installed expansion cards. For switch-configurable boards, the configuration files can be used to determine the proper assignment of resources and to instruct the user about the proper selection of switch settings.

To accomplish the automatic configuration of the system board and expansion cards, EISA uses slot-specific I/O port ranges. An EISA card using these ranges can be installed into any slot in the system without the risk of I/O range conflicts. These I/O ranges can be used for expansion card initialization or for normal I/O port assignments that are guaranteed not to conflict with any other expansion board installed in the system.

EISA also includes a product identification mechanism for system boards and EISA expansion cards. The product identifier allows products to be identified during the configuration and initialization sequences for the system and EISA expansion boards. EISA includes guidelines for selection of a product identifier. The identifier for each product is selected by the product manufacturer and does not need the approval of any other party in the industry. However, a manufacturer-specific ID is assigned to each vendor by BCPR Services, the firm that manages the EISA specification.

EISA Feature/Benefit Summary

Table 1-1 provides a summary of the key features and benefits of the Extended Industry Standard Architecture.

Table 1-1. EISA Feature/Benefit Summary

Feature	Benefit
Backward compatible with all ISA expansion boards	Customer base retains value of installed ISA cards.
Board size	63 square inches of board space permits implementation of powerful, highly-integrated expansion cards.
+5Vdc at approximately 4.5A available at each expansion slot	Ample power for expansion cards employing a large amount of highly-integrated logic.
32-bit address and data buses	Support for 4GB of memory and 32-bit transfers.
Programmable level- or edge-triggered interrupt recognition	Interrupt request lines may be shared by multiple devices.
Enhanced DMA capabilities	Both ISA and EISA DMA devices have access to memory above 16MB. New bus cycle types and 32-bit data bus allow faster transfer speeds (rates of up to 33 MB/second).
Bus Master Support	Support for up to fifteen bus master expansion cards, fast burst bus transfers, automatic data bus steering and control line translation.
Automatic system configuration	Supports automatic configuration of the EISA system board and EISA expansion cards each time the system is powered up. Also provides help to the end user in configuring older ISA expansion cards.

Chapter 2

The Previous Chapter

The previous chapter provided an overview of the features and benefits realized in the EISA environment.

This Chapter

This chapter introduces the EISA bus structure and its relationship to the system board and expansion cards. The concepts of master and slave are introduced and defined. The types of bus masters and slaves are identified.

The Next Chapter

The next chapter, "EISA Bus Arbitration," describes the bus arbitration mechanism implemented in all EISA systems.

Community of Processors

The signals provided on each EISA expansion connector can be divided into four basic categories:

- Address bus group
- Control bus group
- Data bus group
- Bus arbitration group

Three of these four signal groups are present on the expansion slots found in IBM PC/XT/AT products and compatible computers. In EISA, the bus arbitration group has been added.

The EISA specification defines the signals found on the expansion connectors, as well as the permissible bus cycle types that can be performed by bus masters and the software protocol that bus masters must use when communicating with each other. It also defines the support logic residing on the system board

and expansion cards that is necessary to support EISA capabilities. Examples would be the system board's Central Arbitration Control (CAC) and the data bus steering logic, which are discussed in subsequent sections.

Limitations of ISA Bus Master Support

The IBM PC XT/AT and compatible products are essentially single-processor systems. They have one microprocessor located on the system board that uses the address, control and data buses to communicate with the various memory and I/O devices found in a system.

The microprocessor on the system board is the bus master most of the time in a PC/XT/AT. It uses the bus to fetch instructions and to communicate with memory and I/O devices when instructed to do so by the currently executing instruction.

Upon occasion, however, devices other than the microprocessor require the use of the bus in order to communicate with other devices in the system. These devices are the DMA controller and the RAM refresh logic. The DMA controller must use the bus to transfer data between I/O devices and memory. The refresh logic must use the bus periodically to refresh the information stored in DRAM memory.

When a device other than the microprocessor (such as the DMA controller or the refresh logic) requires the use of the bus, it must force the microprocessor to give up control of the bus. This is accomplished by asserting the microprocessor's HOLD (Hold Request) input. Upon detecting HOLD asserted, the microprocessor electrically disconnects itself from the address, control and data buses so the requesting device can use them to communicate with other devices. This is called “floating” the bus. The microprocessor then asserts its HLDA (Hold Acknowledge) output, informing the requesting device that it has yielded the bus, making it the new bus master. The device remains bus master as long as it keeps the microprocessor's HOLD input asserted.

When a bus master other than the microprocessor on the system board has completed using the bus, it deasserts the microprocessor's HOLD input, allowing the microprocessor to re-connect itself to the bus and to become bus master again.

Although it is possible for an expansion card inserted into an IBM PC/XT/AT expansion slot to become bus master, there is a major drawback. When an expansion card becomes bus master in a PC/XT/AT, it can remain bus master as

long as it keeps the microprocessor's HOLD line asserted. There are no safety mechanisms built into a PC/XT/AT to prevent a bus master card from monopolizing the use of the bus to the exclusion of the microprocessor and the RAM refresh logic on the system board and potential bus master cards installed in other expansion connectors.

If, due to poor design or a failure, a bus master expansion card monopolizes the bus for an inordinate amount of time, the main microprocessor cannot continue to fetch and execute instructions. This could have serious consequences. In addition, the refresh logic is unable to become bus master on a timely basis and data in DRAM memory could be lost. Finally, other bus master cards are not able to become bus master and transfer data. To summarize, severe problems can be incurred when bus master expansion cards are used in a PC/XT/AT.

EISA Bus Master Support

The ISA bus mastering problem is fixed in the EISA environment by the addition of the EISA bus arbitration signals and a Central Arbitration Control (CAC) on the EISA system board. The CAC provides a method for resolving situations where multiple bus masters are competing for the use of the bus. As explained in the chapter entitled "EISA Bus Arbitration," a bus master is not permitted to monopolize the bus in an EISA machine.

By establishing a method for resolving bus conflicts, EISA creates a system that can safely support multiple bus masters. This means that EISA products support use of the bus by:

- The microprocessor
- The DMA controller on the system board
- The refresh logic on the system board
- Bus master cards installed in expansion connectors

Typically, a bus master card is quite intelligent, incorporating a microprocessor and its own local ROM, RAM and I/O devices. An example would be a disk controller card built around an 80386 microprocessor, executing its own software from its local (on-board) ROM memory. It stores data received from main memory in its local memory prior to writing it to disk. It can read large amounts of data from disk, store it in its local memory and forward it to another device, such as memory on the system board, when necessary. It controls an array (group) of eight disk drives.

Other bus masters could issue high-level commands or requests to the example disk controller. An example would be a request sent to the disk controller card to search for a database file called "TOM.DBF" on the eight disk drives it controls and, if found, read a particular record and send it back to the requesting bus master. After issuing the request to the disk controller card, the requesting bus master would surrender the EISA bus and continue other local processing until the disk controller card responds. Upon completing the search, the disk controller card would become bus master and transfer the requested data into system memory for the other bus master to use.

An EISA system can safely incorporate a number of intelligent bus master cards, each essentially running on its own. When required, they can communicate with each other and transfer data between themselves either directly or through system memory. The EISA system is designed to support multiprocessing — multiple processors, each handling a portion of the overall task. Properly implemented, the parallel processing accomplished in this type of system is extremely efficient.

Figure 2-1 illustrates the EISA system bus structure. The basic system components are:

- System board
- ISA/EISA expansion cards

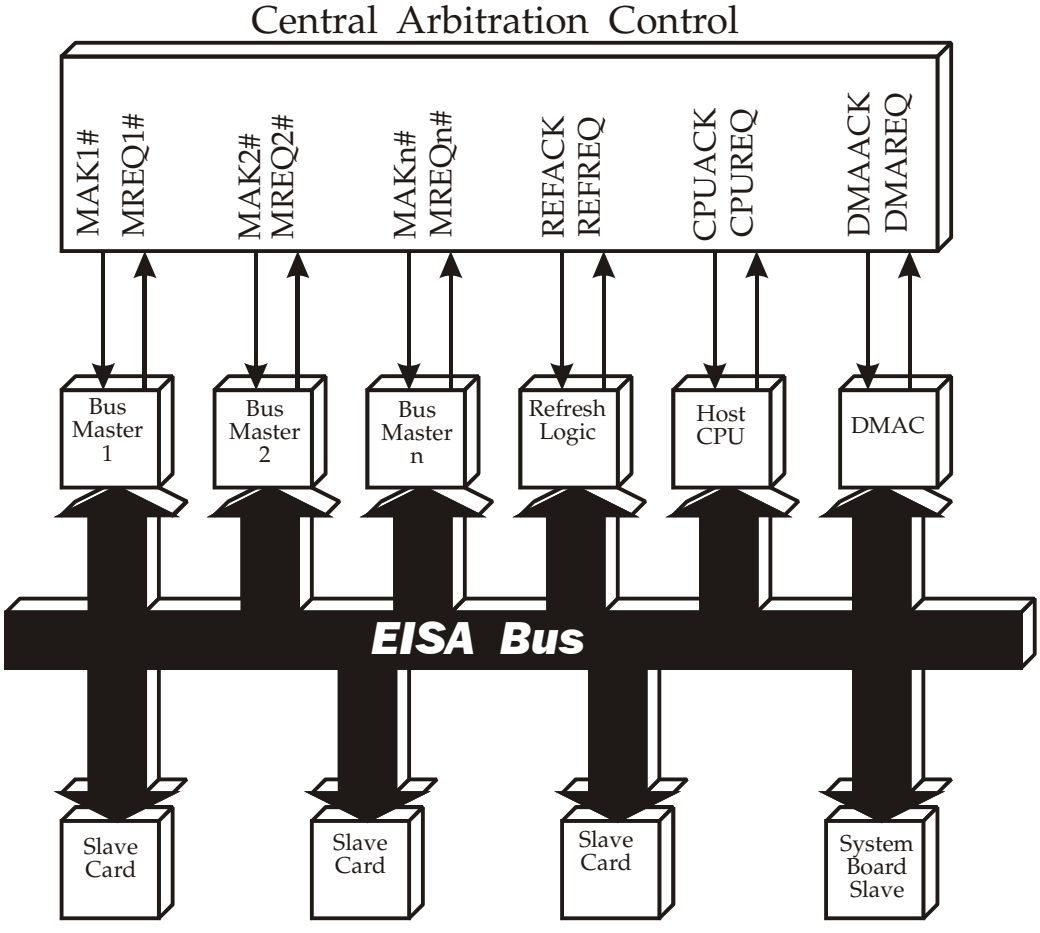


Figure 2-1. The EISA Bus — a Shared Resource

The user may install two basic classes of devices on the expansion bus:

- ISA-compatible expansion cards
- EISA-compatible expansion cards

All EISA and ISA expansion devices fall into one of two categories:

- A **master** is a device that executes bus cycles to communicate with other devices. Any type of master can communicate with any type of slave in the system. The system board provides data bus steering logic that copies the

data between data paths and translates EISA/ISA control signals when necessary.

- A **slave** is a device that a master reads from or writes to. A slave may be either a memory or an I/O slave.

There is only one type of ISA master — the ISA 16-bit bus master. This is a device that attaches to the ISA Bus and is capable of executing bus cycles to communicate with memory or I/O slaves. This is accomplished by interfacing the bus master card to DMA channel five, six or seven with the channel programmed to operate in cascade mode. A more detailed description of bus mastering in the ISA environment can be found in the chapter entitled "DMA and Bus Mastering" in the MindShare book entitled *ISA System Architecture*.

EISA System Bus Master Types

In an EISA system, there are five basic types of bus masters:

- **16-bit ISA or EISA bus master —** This is a 16-bit ISA or EISA device that attaches to the EISA bus and is capable of executing bus cycles to communicate with any slave. When communicating with a 32-bit EISA slave or an 8-bit ISA slave, the data bus steering logic on the system board must sometimes aid in the transfer.
- **32-bit EISA bus master —** This is a 32-bit device that attaches to the EISA bus and is capable of executing bus cycles to communicate with any slave. When communicating with 8 or 16-bit slaves, the data bus steering logic on the system board must sometimes aid in the transfer.
- **Main CPU —** The CPU may communicate with any ISA or EISA Slave or with devices resident on the CPU's local bus structure. When the microprocessor attempts to perform a transfer utilizing one or more data paths not connected to the target slave, the data bus steering logic on the system board must aid in the transfer.
- **The refresh logic —** Used to refresh DRAM memory throughout the system.
- **DMA controllers —** Used to transfer information between an I/O device and system memory.

Types of Slaves in EISA System

Slaves fall into the following categories:

- 8-bit ISA I/O and memory slaves
- 16-bit ISA I/O and memory slaves
- 16-bit EISA I/O and memory slaves
- 32-bit EISA I/O and memory slaves
- 8, 16 or 32-bit slaves on the CPU's local bus

Chapter 3

The Previous Chapter

The previous chapter provided background on ISA's inability to support multiple processors in a fair fashion and introduced the EISA bus and the role of the Central Arbitration Control logic on the EISA system board. The types of bus masters and slaves were identified.

This Chapter

The bus arbitration scheme used by the EISA Central Arbitration Control is described in detail.

The Next Chapter

The next chapter, "Interrupt Handling," describes the methods used to detect and service interrupt requests in both the ISA and EISA environments.

EISA Bus Arbitration Scheme

All EISA systems incorporate a device known as the Central Arbitration Control (CAC) on the system board. The CAC's task is to arbitrate among the outstanding requests for bus ownership and to then grant the bus to the winner.

There are four classifications of devices that can issue requests to the CAC:

- Main CPU
- Expansion bus masters
- Refresh controller on the system board
- DMA Controller (DMAC) on the system board

Figure 3-1 illustrates the CAC's relationship to potential bus masters.

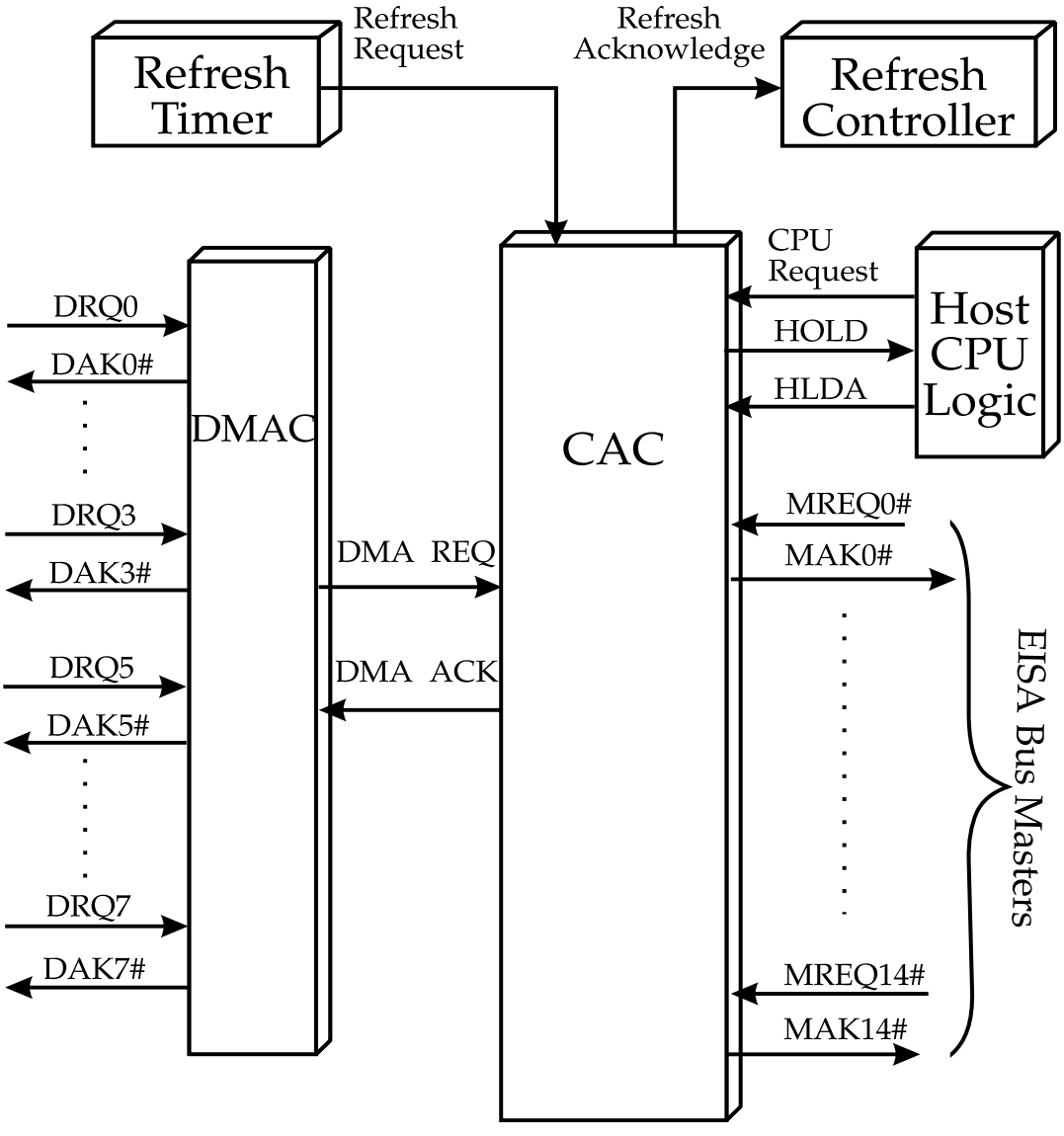


Figure 3-1. Block Diagram of the Central Arbitration Control (CAC)

The CAC uses a multi-level, rotating priority arbitration scheme. Figure 3-2 depicts this rotational priority scheme. On a fully loaded bus, the order in which devices are granted bus access is independent of the order in which they generate bus requests, since devices are serviced based on their position in the ro-

tational order. The DMAC is given a high order of priority to assure compatibility with ISA expansion boards that require short bus latency (because they don't have any buffering). The EISA bus masters are assigned a lower priority and designers of EISA bus master cards must therefore provide for longer bus latency by incorporating buffers.

The top priority level uses a 3-way rotation to grant bus access sequentially to a DMA channel, the refresh controller, and a device from the 2-way rotation (either the main CPU or a bus master card). A DMA channel, the refresh controller and a device from the 2-way rotation each gain access to the bus at least one of every three arbitration cycles (depending on what devices are requesting service). A device that does not request the bus is skipped in the rotation. The main CPU is allowed to retain control of the bus when no other devices are requesting bus mastership. In systems that provide the main CPU with a look-through cache controller, the host processor only requires the use of the bus under the following conditions:

- a cache read miss
- an I/O read or write

In a system wherein the main CPU doesn't have a cache (or uses a look-aside cache), the main CPU frequently requests the use of the bus.

The DMA controller is programmed during the POST to use a fixed priority scheme in evaluating which DMA channel to service next. As pictured in figure 3-2, this means that DMA channel zero has the highest priority, followed by channels two – seven. It should be noted that DMA channel four is unavailable because it is used to cascade the slave DMA controller through the master (see the chapter entitled "DMA and Bus Mastering" in the MindShare book entitled *ISA System Architecture)*.

NMI interrupts are given special priority (because NMI is used to report critical errors). When an NMI interrupt occurs, the arbitration mechanism is modified so that the bus master cards and the DMACs are bypassed each time they come up for rotation. This gives the CPU complete control of the bus for NMI servicing.

DMA priorities can be modified by programming the DMAC control register to use rotating priority. This scenario is pictured in figure 3-3. Each DMA channel then has essentially the same priority as all of the others.

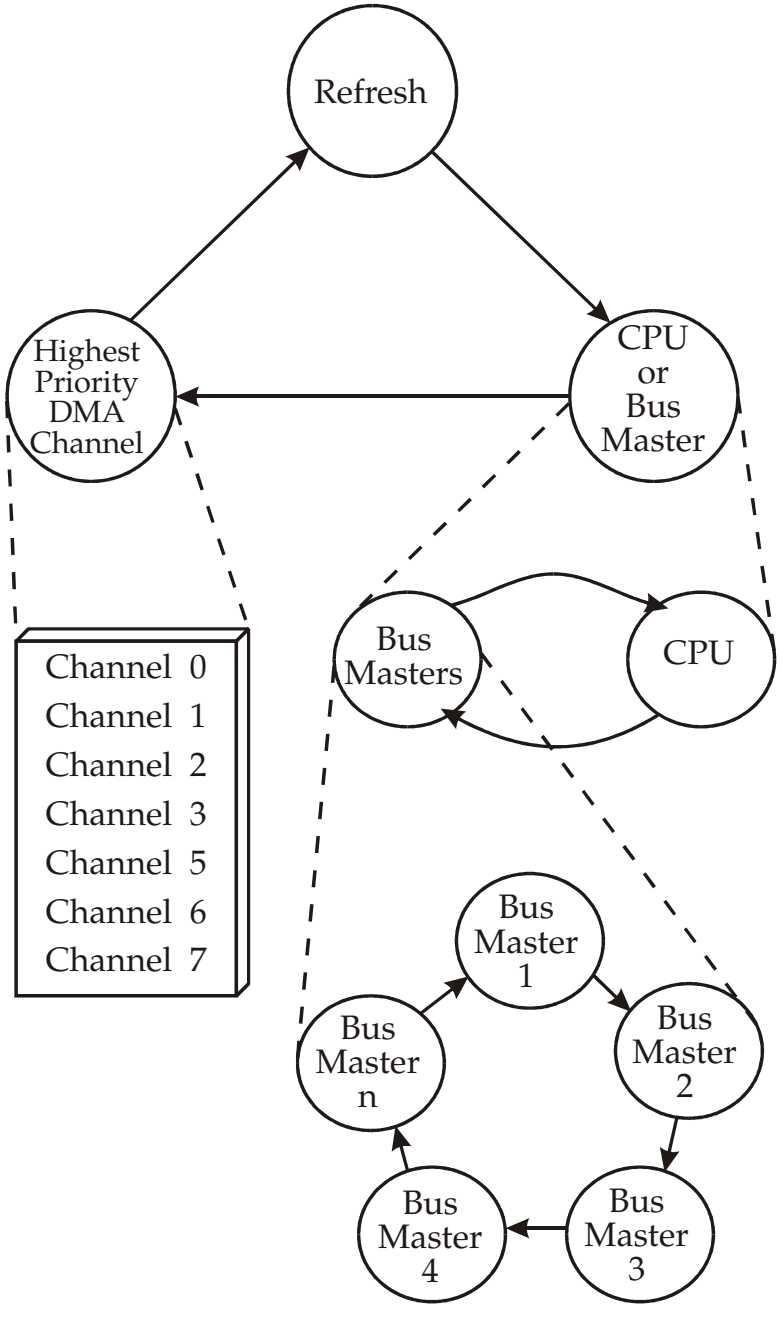


Figure 3-2. CAC with DMACs Programmed for Fixed Priority

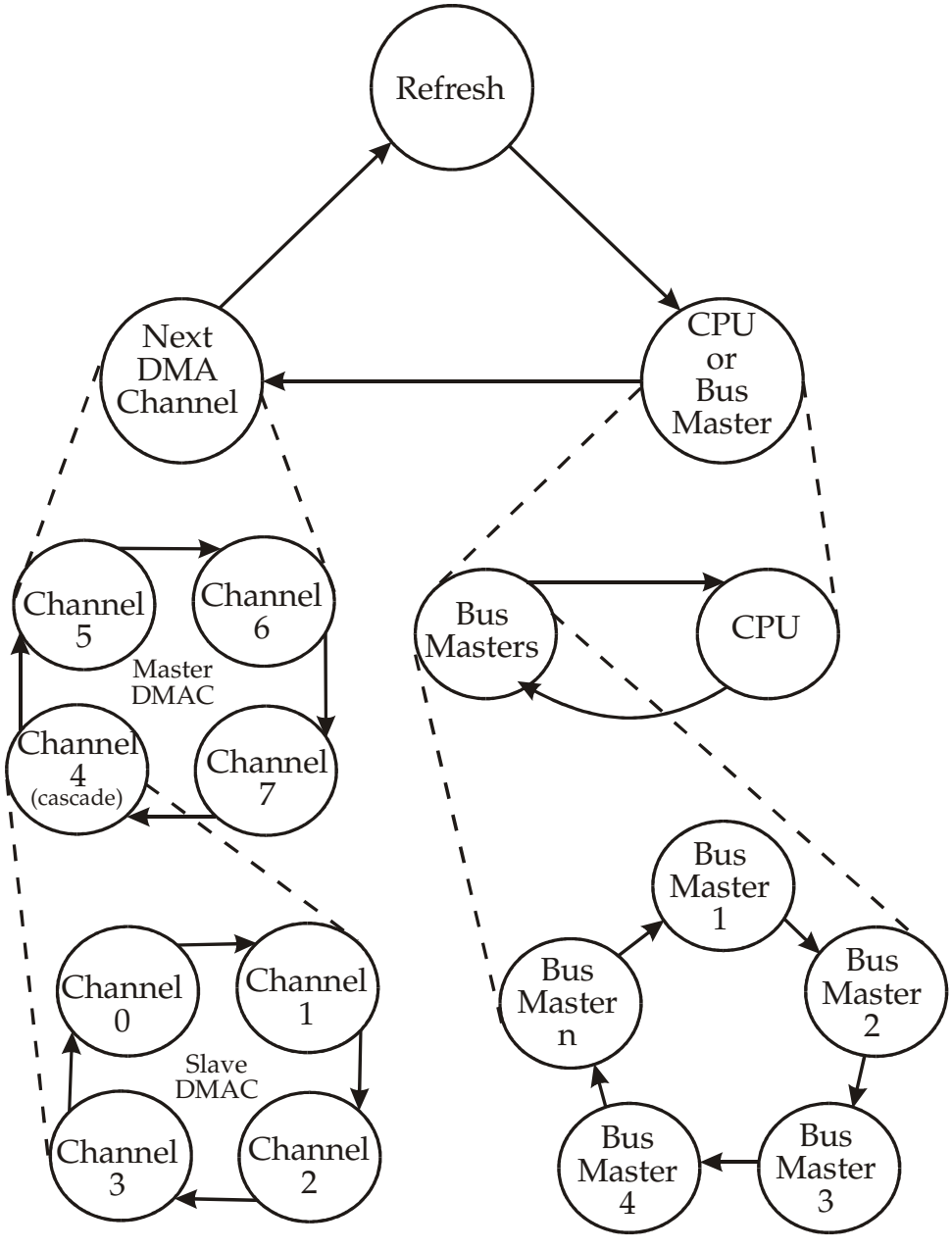


Figure 3-3. CAC with DMACs Programmed for Rotational Priority

Preemption

When one of the bus masters requires the use of the bus to communicate with another device, it must assert its request line (MREQn#) to the CAC (refer to figure 3-1). After deciding which device currently requesting the bus is next in the rotation, the CAC asserts the acknowledge line (MAKx#) associated with the bus master that currently owns the bus. In this way, the CAC preempts the current bus master, commanding it to relinquish control of the bus. Upon being preempted by removal of its acknowledge, the current bus master must relinquish control of the bus within a prescribed period of time. If the current bus master is an EISA bus master card, it must release the bus within 64 cycles of the bus clock signal (BCLK). Since BCLK has a nominal frequency of 8MHz, or 125ns per cycle, 64 BCLK cycles equates to eight microseconds. If the current bus master is a DMA channel programmed for one of the new EISA bus cycle types (rather than the ISA-compatible bus cycle), the DMA channel has 32 BCLKs , or four microseconds, to release the bus. DMA channels programmed to run ISA-compatible DMA bus cycles cannot be preempted. Care should therefore be taken when utilizing an ISA DMA channel to perform a block data transfer using either block or demand transfer modes. If the transfer is too long, other devices requiring the use of the bus, such as the refresh controller, may be forced to wait too long.

The current bus master indicates that it is relinquishing control of the bus by de-activating its CAC request line (MREQx#). If it doesn't relinquish control within eight microseconds, the CAC takes the following actions:

- asserts the reset signal on the EISA bus to force the current bus master off the bus
- asserts NMI to alert the main microprocessor that a bus timeout has occurred
- grants the bus to the main CPU so that it can respond to the NMI

If, on the other hand, the current bus master honors the preemption, relinquishing the bus and deasserting its request to the CAC, the CAC then grants the bus to the next bus master in the rotation that is requesting the use of the bus.

As illustrated in figure 3-1, the main CPU, refresh logic and the DMA controller each have a pair of request/acknowledge lines connecting it to the CAC. In addition, there is also a pair of request/acknowledge lines connected to each EISA connector in the system. The EISA specification provides support for up to fifteen EISA bus masters, numbered from zero to fourteen. MREQ0# and

MAK0# are typically used to implement an EISA-style bus master that is embedded on the system board. MREQ1# and MAK1# are connected to EISA expansion connector one, MREQ2# and MAK2# to EISA connector two, etc. It should be noted, however, that the CAC encapsulated in the Intel 82350DT EISA chip set only has six pairs of EISA request/acknowledge lines and can therefore only support EISA bus master cards in a maximum of six EISA card connectors. This explains why some EISA machines with more than six EISA slots only support bus master cards in six of them.

If the current bus master is preempted during a multiple bus cycle transfer, it will give up the bus as described above, and, after waiting two BCLKs, it reasserts its request line to request the use of the bus again.

Example Arbitration Between Two Bus Masters

The timing diagram in the figure 3-4 illustrates bus arbitration between two Bus masters.

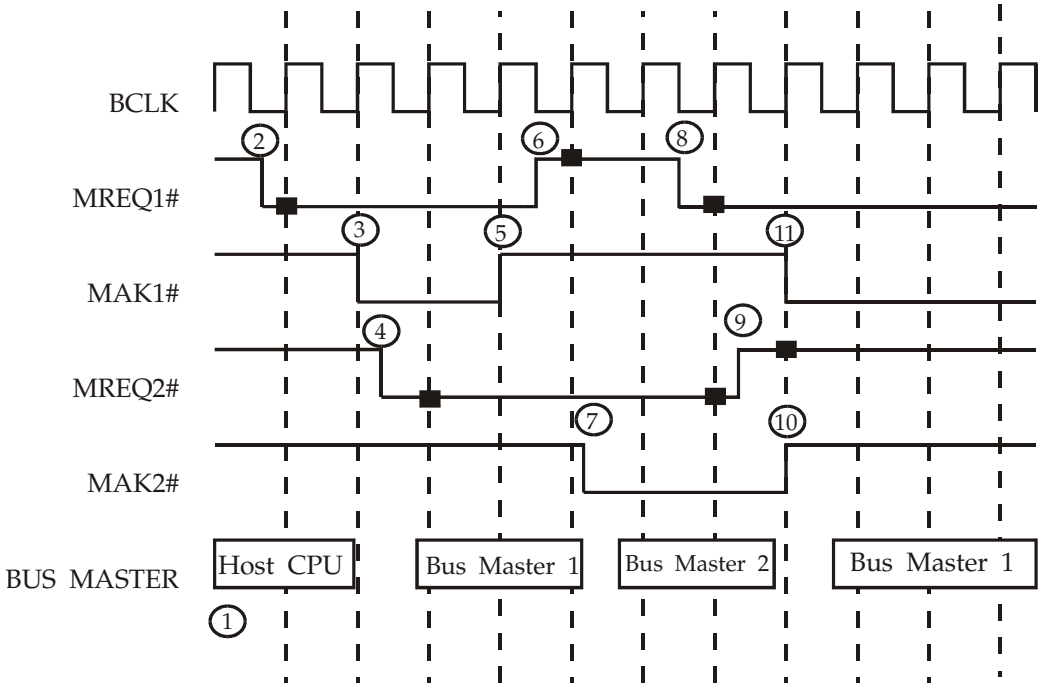


Figure 3-4. Arbitration between Two Bus Masters

The following steps define the sequence of events illustrated in figure 3-4. The step numbers correspond to the reference numbers in the illustration.

1. Initially, the main processor owns the bus.
2. The bus master in slot one requests the use of the bus by asserting MREQ1# (Master Request, slot 1) to the CAC.
3. After the CAC has removed ownership of the bus from the main processor and the main processor signals its willingness to give up ownership, the CAC grants ownership to bus master one by asserting MACK1# (Master Acknowledge, slot 1). Bus master one now owns the bus and can initiate one or more bus cycles.
4. The bus master in slot two signals its request for bus mastership by asserting MREQ2# to the CAC.
5. The CAC signals bus master one that it must give up bus mastership by removing MACK1#.
6. After detecting its MACK1# deasserted, bus master one has up to eight microseconds to release the bus. This gives it time to complete one or more bus cycles prior to release. Bus master one signals its release of the bus by deasserting MREQ1#.
7. The bus is granted to bus master two by the CAC when it asserts MACK2#.
8. Bus master one requires the use of the bus again to either complete its previously-interrupted series of transfers or to initiate a new transfer. It signals its request to the CAC by asserting MREQ1#.
9. Bus master two has finished using the bus, so it voluntarily gives up ownership by deasserting MREQ2#.
10. The CAC removes ownership from bus master two by deasserting MACK2#.
11. The CAC grants the bus to bus master one again by asserting MACK1#.

Memory Refresh

The EISA system board incorporates a refresh controller that requests the use of the bus once every fifteen microseconds to refresh a row of DRAM memory. 16-bit ISA bus masters that hold the bus longer than fifteen microseconds must perform memory refresh bus cycles at the fifteen microsecond interval.

The EISA refresh controller includes a 14-bit row counter that drives its contents onto address lines 15:2 when the refresh controller becomes bus master. The refresh controller also places a value on BE#[3:0] to be transferred to A[1:0] and SBHE#.

Each time that the refresh controller requests the use of the bus and the request is not granted within fifteen microseconds, the refresh controller increments its uncompleted refresh count. This counter can count up to four uncompleted refresh bus cycles. When the refresh controller succeeds in gaining control of the bus, it performs a refresh bus cycle and decrements the uncompleted refresh count by one. If more refreshes are queued up (the count isn't exhausted), the refresh controller immediately requests the use of the bus again without waiting the normal period of fifteen microseconds.

Chapter 4

The Previous Chapter

The previous chapter described the bus arbitration scheme utilized in EISA machines.

This Chapter

An in-depth discussion of interrupt request handling in the ISA environment can be found in the chapter entitled "Interrupt Handling" in the MindShare book entitled *ISA System Architecture*. This chapter provides a brief review of the ISA interrupt request handling method and a detailed description of the EISA method.

The Next Chapter

The signals and support logic that comprise the ISA bus impose certain limitations on performance and capabilities. The EISA specification builds upon the ISA bus, adding new bus signals and system board support logic. The end result is backward-compatibility with all ISA cards and improved performance and capabilities for EISA cards. The next chapter provides a detailed description of the extensions to the ISA bus and support logic.

ISA Interrupt Handling Review

The Intel 8259 interrupt controller's interrupt request inputs can be programmed to recognize either a rising-edge or a static high level as a valid interrupt request. The programmer may select either of these recognition modes for all eight inputs at once. There is no provision for the selection of either type on an input-by-input basis. On an ISA machine, the 8259 interrupt controllers are programmed to recognize a rising-edge as a valid interrupt request on its eight inputs. The following section describes the two shortcomings inherent in ISA interrupt handling.

ISA Interrupt Handling Shortcomings

Phantom Interrupts

Internally, the 8259 has a pull-up resistor on each of its IRQ inputs. When an ISA expansion card must generate an interrupt request, the line is driven low by the card and is then allowed to go high again. The low-to-high transition is registered as an interrupt request by the 8259 interrupt controller on the system board. The 8259 specification also demands that the IRQ line must remain high until after the leading-edge of the first interrupt acknowledge bus cycle generated by the host processor. The pull-up resistor ensures that this will be the case.

Consider the case where an ISA card is designed to keep its IRQ line low until a request must be generated. At that time, the card would allow the IRQ line to go high and would maintain the high until the request has been serviced. The transition from low-to-high would be registered as a request by the 8259. When the request has been serviced, the card would drive the line low again and keep it low until the next request is to be generated. Although this design would work, a problem may arise.

A transitory noise spike on this interrupt request line could register as a valid interrupt request. When the microprocessor issues the first of the two interrupt acknowledge bus cycles, however, the IRQ line will already be low again. This means that the IRQ line's respective IRR (Interrupt Request Register) bit will not be active. The first interrupt acknowledge resets the highest-priority IRR bit and sets its associated bit in the ISR (In-Service Register). In this case, since the IRR bit is no longer set because the request was of too short a duration (a ghost, or phantom, interrupt), the 8259 must take special action. The 8259 is designed to automatically return the interrupt vector for its number seven input in this case. When the microprocessor then generates the second interrupt acknowledge, the 8259 sends back the vector associated with its number seven input. On the system's master 8259, this is 0Fh, the vector of IRQ7. On the slave, it is 77h, the IRQ15 interrupt vector. The microprocessor therefore jumps to either the IRQ7 or the IRQ15 interrupt service routine.

In these two routines, therefore, the programmer must perform a check to see if the IRQ7 or the IRQ15 was real. This is accomplished by reading the contents of the respective 8259's ISR register and checking to see if bit seven is really set. If it is, then the request is real and the programmer should execute the remainder of the interrupt service routine to service the request. If, on the other hand, the

bit is cleared, it was a phantom or ghost interrupt and the programmer should execute an interrupt return (the IRET instruction) to return to the interrupted program flow.

ISA card designers can avoid this problem by designing the card's IRQ output driver to remain tri-stated when not requesting service, allowing the pull-up resistor inside the 8259 to keep it high. The line is not prone to pick up noise spikes when it's high. When a request must be generated, the card drives the IRQ line low and then lets it go high again. This low-to-high transition registers as a request in the 8259. When the microprocessor generates the first interrupt acknowledge, the line is guaranteed to be high and the request is therefore valid.

Limited Number of IRQ Lines

In the ISA environment, IRQ lines are not shareable because only one transition is registered if more than one card generates a transition. The low-to-high transition generated by the first card is recognized by the interrupt controller and any subsequent transitions are ignored until the first request has been serviced. More than one ISA device may share an IRQ line as long as it is guaranteed that they never generate requests simultaneously. Since only one device may use each IRQ line, a fully-loaded machine may easily use up all of the available lines. An in-depth discussion of interrupt handling in the ISA environment may be found in the MindShare book entitled *ISA System Architecture*.

EISA Interrupt Handling

Shareable IRQ Lines

The interrupt controllers used in the EISA environment are a superset of the Intel 8259A controller. The 8259A allows the programmer to gang-program all eight IRQ inputs as either edge or level-triggered. In the EISA environment, machines must be capable of supporting both ISA and EISA-style cards. This means that the programmer must have the ability to individually select each IRQ input as either edge or level-triggered. The EISA interrupt controller has added an additional register for this purpose.

The ELCR, or Edge/Level Control Register, provides this selectivity. The master interrupt controller's ELCR is located at I/O address 04D0h, while the slave's is at I/O address 04D1h. Each of these registers is default programmed

to edge-triggering upon power-up. Tables 4-1 and 4-2 illustrate their respective bit assignments.

Table 4-1. Master Interrupt Controller's ELCR Bit Assignment

Bit	Description
7	0 = IRQ7 is edge-sensitive and non-shareable, 1 = IRQ7 is level-sensitive and shareable.
6	0 = IRQ6 is edge-sensitive and non-shareable, 1 = IRQ6 is level-sensitive and shareable.
5	0 = IRQ5 is edge-sensitive and non-shareable, 1 = IRQ5 is level-sensitive and shareable.
4	0 = IRQ4 is edge-sensitive and non-shareable, 1 = IRQ4 is level-sensitive and shareable.
3	0 = IRQ3 is edge-sensitive and non-shareable, 1 = IRQ3 is level-sensitive and shareable.
2	always 0 because the master's IRQ2 input is used to cascade the slave interrupt controller's output through the master.
1	IRQ1 is dedicated to the interrupt request output of the keyboard interface. This bit must be 0, selecting edge-sensitive and non-shareable.
0	IRQ0 is dedicated to the interrupt request output of the system timer. This bit must be 0, selecting edge-sensitive and non-shareable.

Table 4-2. Slave Interrupt Controller's ELCR Bit Assignment

Bit	Description
7	0 = IRQ15 is edge-sensitive and non-shareable, 1 = IRQ15 is level-sensitive and shareable.
6	0 = IRQ14 is edge-sensitive and non-shareable, 1 = IRQ14 is level-sensitive and shareable.
5	IRQ13 is dedicated to the error output of the numeric coprocessor. This bit must be 0, selecting edge-sensitive and non-shareable. In reality, IRQ13 is shared by the numeric coprocessor and the chaining interrupt output of the DMA controller. More information regarding chaining can be found in the chapter entitled "EISA DMA."
4	0 = IRQ12 is edge-sensitive and non-shareable, 1 = IRQ12 is level-sensitive and shareable.
3	0 = IRQ11 is edge-sensitive and non-shareable, 1 = IRQ11 is level-sensitive and shareable.
2	0 = IRQ10 is edge-sensitive and non-shareable, 1 = IRQ10 is level-sensitive and shareable.
1	0 = IRQ9 is edge-sensitive and non-shareable, 1 = IRQ9 is level-sensitive and shareable.
0	IRQ8 is dedicated to the alarm output of the Real-Time Clock chip. This bit must be 0, selecting edge-sensitive and non-shareable.

When programmed to recognize level-sensitive interrupt requests, the interrupt controller recognizes a low on an IRQ line as a request and the interrupt request line may be shared by two or more devices. The following paragraphs define how this works.

During the POST, software scans the area of memory space set aside for device ROMs, typically C0000h – DFFFFh, to determine if any expansion cards have device ROMs. When a device ROM is detected, the POST jumps to the initialization routine in the ROM to execute the card's POST and to install the start addresses of its interrupt service and BIOS routines into the proper entries in the interrupt table in memory. The ROM's initialization routine reads the current pointer from the interrupt table entry, saves it and writes the pointer to the device ROM's interrupt service routine in its place. After testing and initializing the card, the ROM code performs a return to the system POST. The POST then continues to scan the device ROM memory area for other device ROMs. If any are found, the same process is repeated. When another card is using the same IRQ line, its ROM code reads the current pointer from the interrupt table entry and saves it. This pointer points to the interrupt service routine within a previously detected device ROM for another EISA I/O card that is sharing this IRQ

line. The second card's ROM code then stores a pointer to its own interrupt service routine into the IRQ line's assigned interrupt table entry. In this way, a linked list of interrupt service routine start addresses is created. Any loadable-device drivers or TSRs that use shareable interrupt request lines should do the same.

Each shareable interrupt request line has a pull-up resistor on it (internal to the EISA interrupt controller). When no request is being generated, or when no I/O devices are physically connected to the line, the line is pulled-up to a high level. This provides a good deal of noise immunity on the line, preventing spurious requests.

Figure 4-1 illustrates how multiple EISA I/O cards can share the same IRQ line. An I/O device that places a low on an interrupt request line when it generates a request may share the line with other devices that use it the same way. When a board of this type must generate a request, it acts as follows:

- Through an open-collector driver, it creates a path to ground. This places a low on the 8259's request input. If other I/O devices are sharing the line and generate requests simultaneously, the shared IRQ line is low.
- When an I/O board generates a request, it should also set its interrupt pending bit in a pre-defined I/O port on the card.

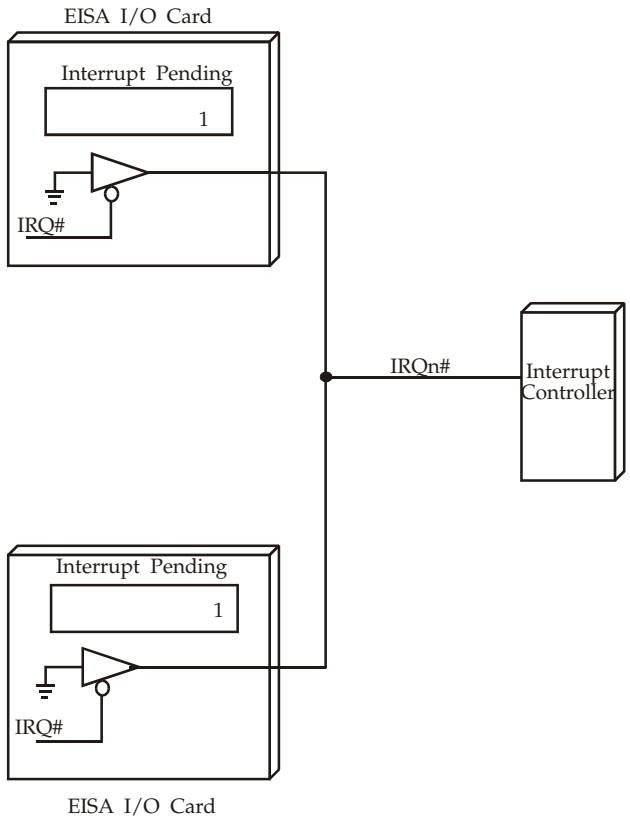


Figure 4-1. IRQ Line Sharing

When the 8259 senses a low on a shareable request input, it generates an interrupt request to the microprocessor. When the microprocessor requests the interrupt vector, the 8259 responds with the vector for the line currently being serviced. The microprocessor then jumps to the interrupt service routine for the last device ROM detected during the POST. In this routine, an I/O read is performed from the card's interrupt pending register to determine if this card is generating a request. If the card's interrupt pending bit is set, the program continues and executes the remainder of the card's interrupt service routine to service the request. If the card's interrupt pending bit isn't set, however, the program jumps to the next interrupt service routine in the chain. The second service routine then polls its respective card's interrupt pending register to determine if it is generating a request. The act of servicing the request (for example, sending a character to a serial port) causes the requesting board's interrupt

pending bit to be cleared. The board also ceases to provide a path to ground for the interrupt request line.

If more than one I/O device were generating requests simultaneously, the other board or boards are still driving a low onto the shared request line. The 8259 would therefore immediately sense another pending request and proceed as outlined above. This time, the interrupt pending bit for the board that was already serviced is determined to be cleared, a jump is executed to the next service routine in the chain to determine if its interrupt pending bit is set.

Since the program must go through a linked service routine list to determine which board(s) is currently generating a request, it stands to reason that the lower down in the list a device is, the more time it will take to service its request (if other devices, further up the list, are also generating requests). This latency, or delay, could cause problems ranging from slow servicing of a device right up to overflow conditions and missing characters. The problem can be solved in one of two ways:

1. Move some devices to other interrupt requests lines.
2. During the configuration process, install the devices requiring the smallest latency first and the others later in the process.

Phantom Interrupt Elimination

All IRQ inputs that are configured as level-sensitive, shareable inputs assume the high state when no requests are pending or when the IRQ line is unused. This renders these inputs relatively noise-free, substantially decreasing the possibility of phantom interrupts.

Chapter 5

The Previous Chapter

The previous chapter provided a detailed description of interrupt handling in the EISA environment.

This Chapter

This chapter provides a description of all the signals on the EISA bus.

The Next Chapter

In the next chapter, the types of bus cycles performed by the main CPU and EISA bus masters are described.

Introduction

The EISA bus consists of two sets of signal lines:

- the ISA Bus
- the extension to the ISA Bus (the EISA bus extension)

Figure 5-1 illustrates the construction of the EISA connector. When installed, ISA boards are physically stopped by the EISA access key and make contact only with the ISA contacts. When an EISA board is installed, however, an alignment notch in the board allows it to bottom out, making contact with both the ISA and the EISA contacts.

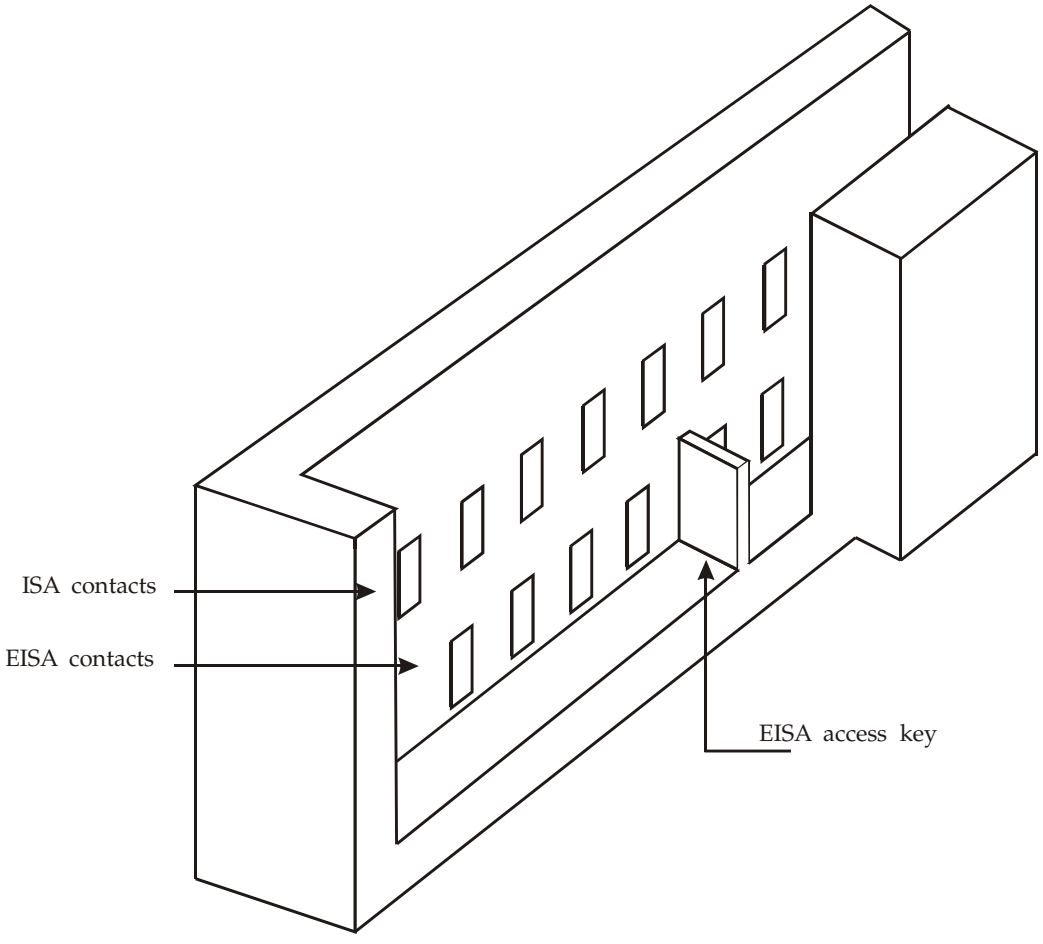


Figure 5-1. The EISA Connector

Many of the ISA signals have already been defined in preceding sections of this book and all of them are fully defined in the MindShare book entitled *ISA System Architecture.* This section is confined to a description of the EISA signals. The following are the signal groups that comprise the EISA Bus.

- Address bus extension
- Data bus extension
- Bus Arbitration signal group
- Burst handshake signal group
- Bus cycle definition signal group

- Bus cycle timing signal group
- Lock signal
- Slave size signal group
- AEN signal

The following paragraphs provide a description of each of these signal groups.

Address Bus Extension

One of the restrictions imposed by the ISA bus structure is a function of the width of the address bus. It consists of 24 address lines, A[23:0]. This permits the microprocessor to address any memory location between address 000000h and FFFFFFh, a range of 16MB.

With the advent of multi-tasking, multi-user operating systems, access to a greater amount of memory became an imperative. The EISA specification expands the address bus to 32 bits (A31:0]), and also adds the byte enable lines, BE#[3:0], to provide 32-bit bus master address support. The ISA bus includes the following address lines:

- SA[19:0]
- LA[23:17]
- SBHE#

The EISA address bus consists of the following signals:

- SA[1:0] (ISA bus)
- SBHE# (ISA bus)
- LA[23:17] (ISA bus)
- LA#[31:24] (EISA extension)
- BE#[3:0] (EISA extension)
- LA[16:2] (EISA extension)

The EISA specification extends the size of the LA Bus to include LA[16:2] and LA#[31:24]. Refer to figure 5-2. Combined with the previously-defined SA bus and LA signal groups on the ISA portion of the bus, this extends the address bus to a full 32-bits, allowing the current bus master to generate any memory address in the range 00000000h – FFFFFFFFh. This is a range of 4GB (giga = billion).

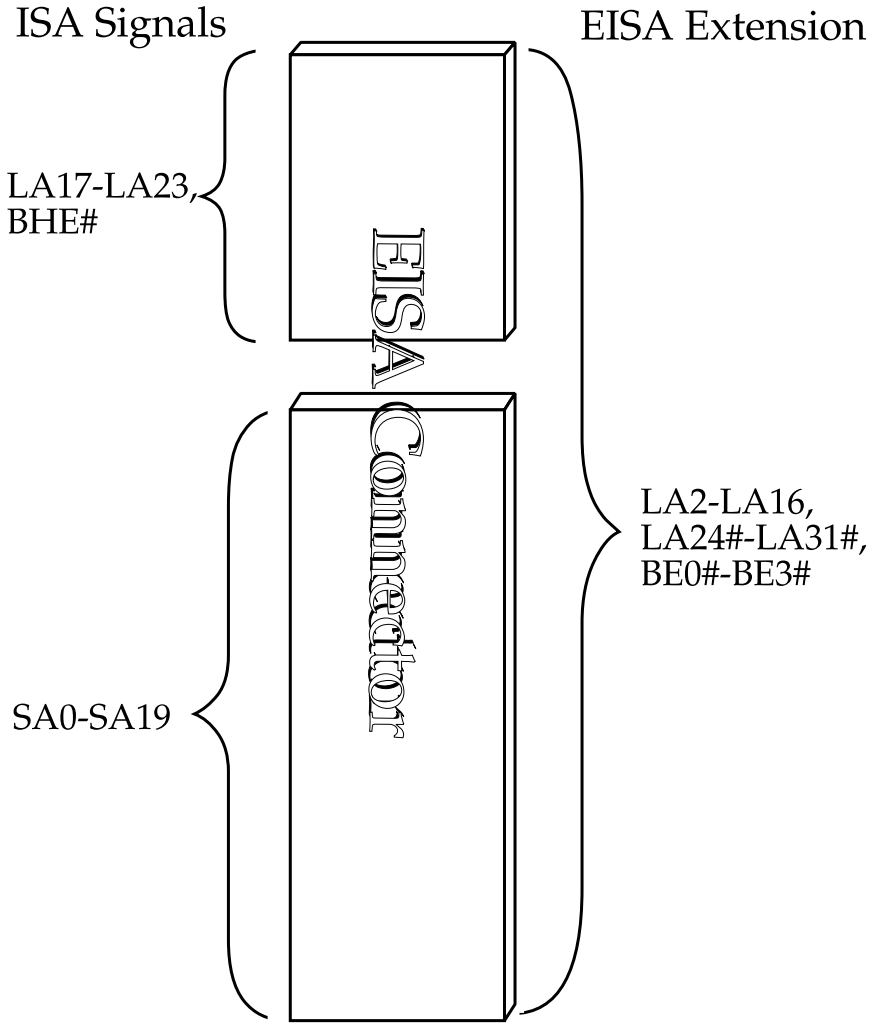


Figure 5-2. The EISA Connector Address Lines

LA#[31:24] are asserted low to prevent 16-bit bus masters from inadvertently selecting 32-bit memory cards residing above 16 MB. When a 16-bit bus master places an address on the address bus, it is only using lines A[23:0]. If address lines LA#[31:24] were allowed to float, a 32-bit memory card that resides above the 16MB boundary might be inadvertently selected. Rather, LA#[31:24] are pulled high with pull-up resistors on the system board, ensuring that they are deasserted unless asserted by a 32-bit bus master. 32-bit EISA memory cards

are designed to recognize that these upper address lines carry inverted address information (0 on a line is a logical 1 and a 1 is a logical 0).

Since the address information on the LA bus shows up sooner than the address on the SA bus (due to address pipelining and the fact that the LA bus bypasses the address latch on the system board), memory cards that use the LA lines can perform an early address decode. This allows the memory card designer to use slightly slow (inexpensive) memory chips and yet achieve higher throughput. In addition, the fact that the LA bus now includes the lower part of the address bus allows memory cards that use SCRAM or Page Mode RAM to determine if the next access will be in the same row of memory (because the row portion of the DRAM address is carried over the lower portion of the address bus).

The EISA specification also adds the four byte enable signal lines, BE#[3:0], allowing 32-bit bus masters to generate addresses in doubleword address format (A[31:2] plus the BE lines) and 32-bit slaves to see the address in 32-bit doubleword format.

Data Bus Extension

The EISA specification extends the width of the data bus by adding two additional data paths consisting of SD[23:16] and D[31:24]. Using these data paths plus the two ISA data paths allows 32-bit bus masters to transfer four bytes (a doubleword) during a single transfer when communicating with 32-bit slaves.

Bus Arbitration Signal Group

Under EISA, two signals have been added to allow implementation of bus master cards. They are described in table 5-1.

Table 5-1. EISA Bus Master Handshake Lines

Signal Name	Full Name	Description
MREQx#	Master Request for slot x	When a bus master in a slot requires the use of the bus to perform a transfer, it asserts its slot-specific MREQx# signal line. This signal is applied to the CAC on the system board, which then arbitrates its priority against other pending bus requests.
MAKx#	Master Acknowledge for slot x	When the CAC is ready to grant the bus to a requesting bus master (MREQx# is asserted), the CAC asserts the bus master's MAKx# slot-specific signal line to inform the bus master that it has been granted the bus.

Figure 5-3 illustrates the relationship of the master request and acknowledge lines to the CAC. The subject of bus arbitration is covered in detailed the chapter entitled "EISA Bus Arbitration."

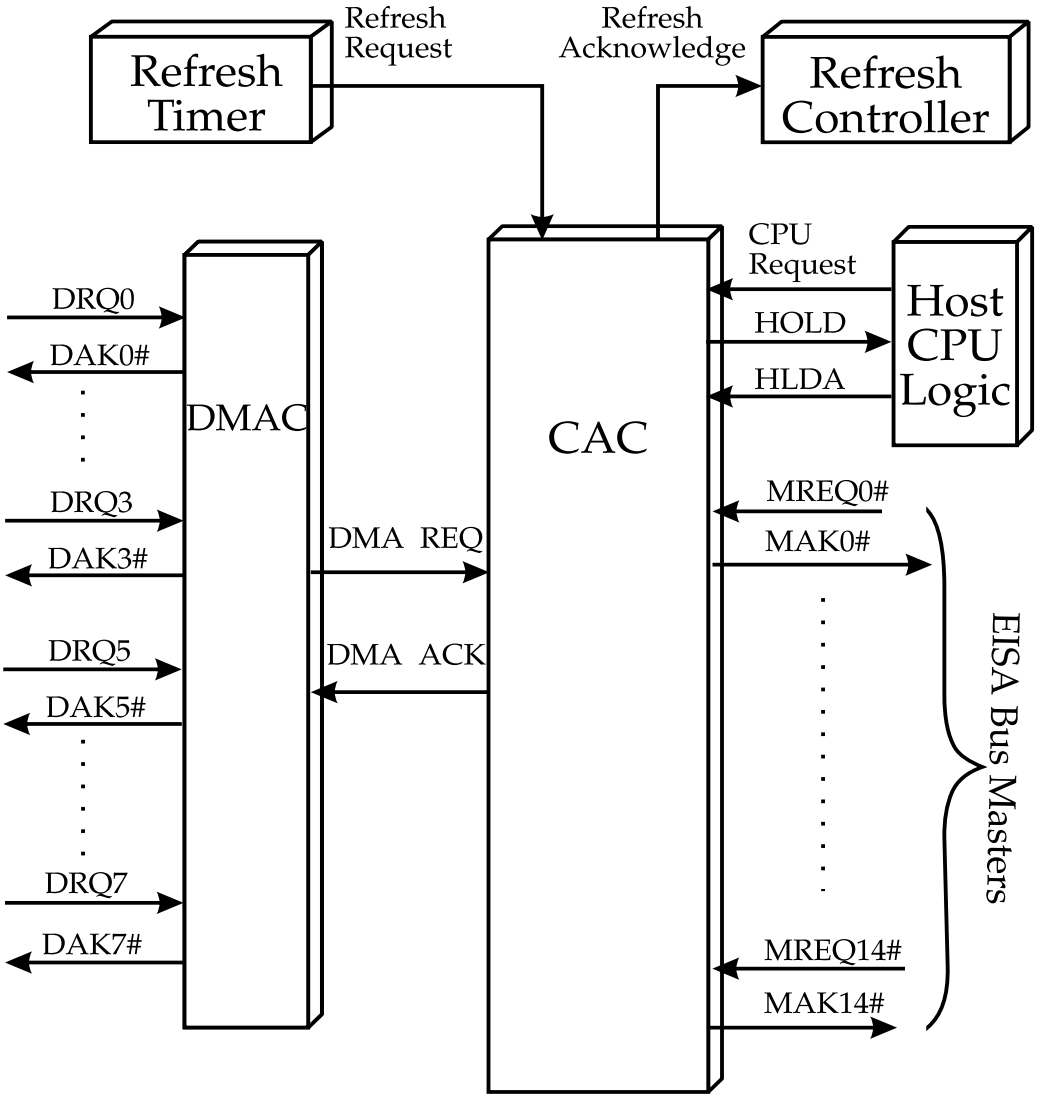


Figure 5-3. The Bus Master Handshake Lines

Burst Handshake Signal Group

The EISA specification adds two signal lines to support initiation of burst mode (Type C) bus cycles. They are described in table 5-2.

Table 5-2. The Burst Handshake Lines

Signal Name	Full Name	Description
SLBURST#	Slave Burst	When addressed, a slave asserts SLBURST# to indicate that it supports burst cycles. If the slave supports burst cycles, it asserts this signal regardless of the state of the MSBURST# signal line.
MSBURST#	Master Burst	During a bus cycle, the current bus master asserts this line as a response to the assertion of SLBURST#. This informs the addressed slave that the bus master supports burst cycles.

The subject of burst mode (Type C") bus cycles is covered in detail in the chapter entitled "EISA CPU and Bus Master Bus Cycles."

Bus Cycle Definition Signal Group

The EISA specification defines a new set of bus cycle definition signal lines. The current EISA bus master uses them to inform the currently addressed slave of the type of bus cycle in progress. Table 5-3 defines the new signals.

Table 5-3. EISA Bus Cycle Definition Lines

Signal Name	Full Name	Description
M/IO#	Memory or I/O	During a bus cycle, M/IO# is set high if a memory address is on the address bus. It is set low if it's an I/O address.
W/R#	Write or Read	During a bus cycle, W/R# is set high if a write bus cycle is in progress and low if a read bus cycle is in progress.

Bus Cycle Timing Signal Group

Under the EISA specification, the signals described in table 5-4 were added to define the address and data portions of the bus cycle, as well as the end of the bus cycle.

Table 5-4. EISA Bus Cycle Timing Signals

Signal Name	Full Name	Description
START#	Start phase	Every EISA bus cycle consists of two phases: the start and command phases. The address and the M/IO# control line are output by the current bus master and decoded by the target slave during the start phase. The start phase corresponds to address time and is therefore one BCLK is duration.
CMD#	Command phase	Every EISA bus cycle consists of two phases: the start and command phases. The data is transferred during the command phase. CMD# is asserted at the trailing edge of the START# signal (trailing edge of Ts) and stays asserted until the end of the bus cycle. When a bus cycle has wait states inserted, the CMD# signal remains asserted for multiple cycles of BCLK.
EXRDY	EISA Ready	Deasserted by an EISA slave to request the insertion of wait states in the current bus cycle. It is sampled on each falling edge of BCLK after the CMD# line is asserted. When sampled asserted, the bus cycle will be terminated at the next rising edge of BCLK.

Lock Signal

The LOCK# signal is asserted by the current bus master to prevent other bus masters from arbitrating for the use of the bus. This allows the current bus master to complete one or more memory accesses prior to surrendering control to another bus master. The purpose of the bus lock capability is to prevent two bus masters that share a memory location as a software semaphore from becoming de-synchronized with each other.

Slave Size Signal Group

When the current bus master addresses an EISA-style slave, the slave asserts one of these two signals to indicate the data paths it can use and to signal that it is an EISA-style slave. Table 5-5 describes these two signals.

Table 5-5. The EISA Type/Size Lines

Signal Name	Full Name	Description
EX32#	EISA Slave Size 32	When a 32-bit EISA slave decodes its address, it asserts EX32# to inform the current bus master that it can handle 32-bit transfers.
EX16#	EISA Slave Size 16	When a 16-bit EISA slave decodes its address, it asserts EX16# to inform the current bus master that it can handle 16-bit transfers.

AEN Signal

The following paragraph describes the manner in which the AEN signal is used under the ISA specification.

When either the master or slave DMA Controller (DMAC) on the system board becomes bus master, it asserts AEN as a substitute for BALE, indicating that a valid memory address is present on the address bus. Memory cards then decode the address on the address bus. I/O cards also monitor the AEN signal line and ignore the address on the bus when AEN is asserted. This is necessary because the DMAC asserts either the IORC# or IOWC# line and I/O devices think that there is an I/O address on the bus when there really isn't.

It should be noted that AEN has another, special, usage in the EISA environment. This additional function is discussed in the chapter entitled "EISA System Configuration."

EISA Connector Pinouts

The EISA connector is an extended version of the ISA connector. The ISA connector is divided into an 8-bit connector and a 16-bit extension. In figure 5-4, the upper half of the EISA connector, rows A and B, comprise the 8-bit portion that is compatible with the IBM PC and XT expansion connector and the 8-bit

portion of the connector found in the IBM PC/AT. On the lower half of the EISA connector in the figure, rows C and D comprise the 16-bit portion that is compatible with the 16-bit extension to the 8-bit connector found in the IBM PC/AT. The pins on the EISA connector are arranged in eight rows. Rows A, B, C, and D comprise the ISA group, while rows E, F, G and H comprise the EISA group.

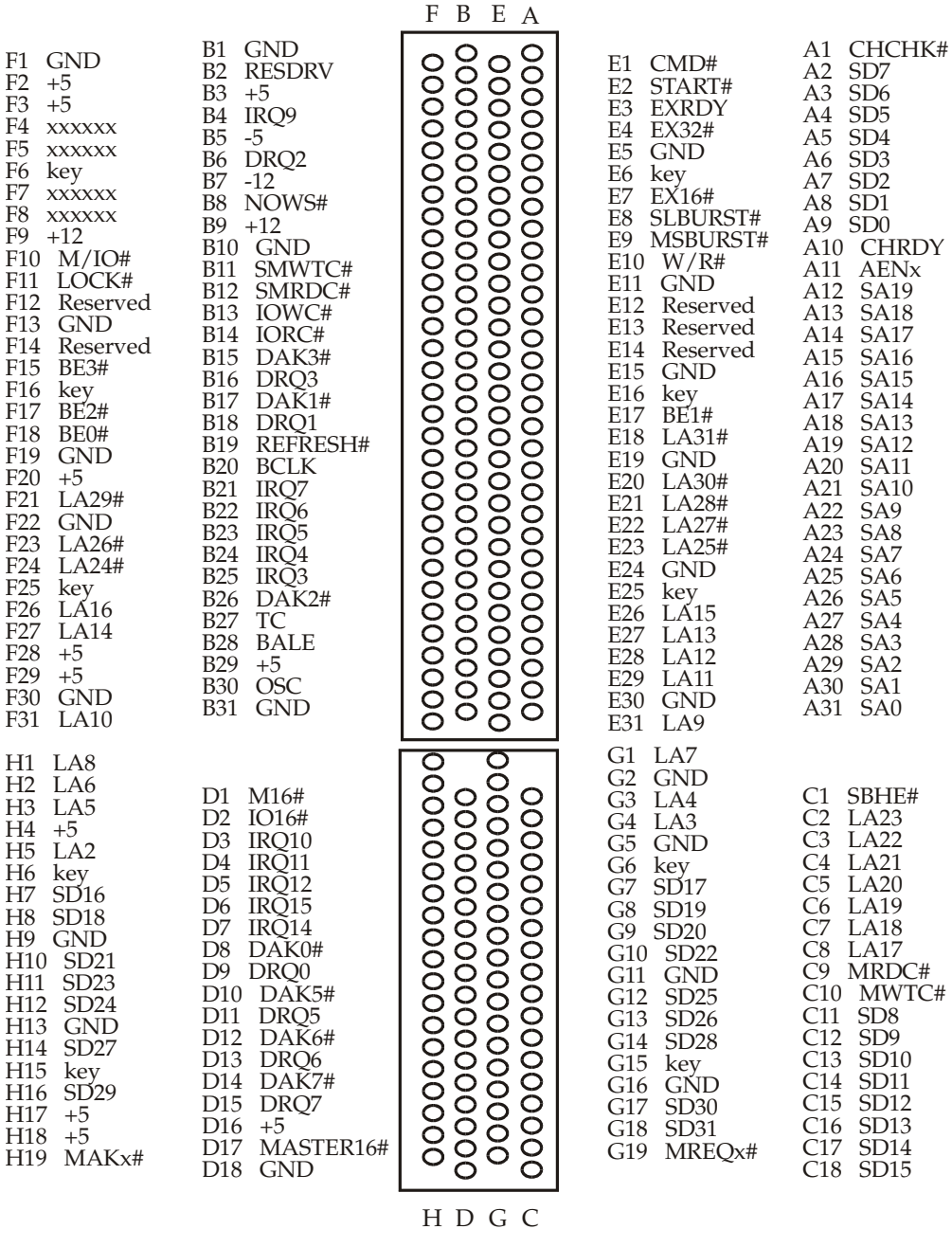


Figure 5-4. The EISA Connector Pin Assignments

Chapter 6

The Previous Chapter

The previous chapter provided a functional description of the EISA bus signals.

This Chapter

This chapter provides a brief review of the ISA bus master and DMA bus cycles. For a detailed description of this subject matter, refer to the MindShare book entitled *ISA System Architecture*.

The Next Chapter

The next chapter provides a detailed description of the EISA CPU and bus master bus cycle types.

Introduction

In order to define extensions to ISA, the writers of the EISA specification had to first document ISA. The following descriptions of ISA bus cycles are based on the descriptions found in the EISA specification.

The Bus Clock (BCLK) is supplied to the ISA bus by the system board and defines the time slots (Tstates) that comprise a bus cycle. In order to maintain ISA compatibility, the maximum clock rate used for bus cycles on the EISA bus is 8.33MHz.

8-bit ISA Slave Device

An 8-Bit ISA slave device interfaces only to the least-significant eight data bus bits and uses only ISA address bus bits SA[19:0]. This is the simplest and slowest of the slave devices and was first developed for use with the IBM PC. These devices didn't have to be very fast because the PC was based on an Intel 8088 microprocessor running at 4.77MHz.

At 4.77MHz, the clock period is 209.64ns and a 0-wait state bus cycle consists of four clock cycles (838.6ns). In other words, devices with an access time of up to 836ns could be interfaced to the microprocessor without incurring any wait states. Since the designer must account for the cycle time of DRAMs (typically double the access time), not the stated access time, this means that DRAMs with an access time of up to 400ns could be interfaced to the microprocessor without incurring wait states.

16-bit ISA Slave Device

A 16-bit ISA slave device interfaces to sixteen ISA data bus bits and uses ISA address bus lines SA[19:0], LA[23:17] and SBHE#. 16-bit devices were developed for use with the IBM PC-AT. Some of these devices were designed to operate in the original 6MHz version of the IBM PC-AT, while most were designed to work with the 8MHz version.

The 6MHz PC-AT could interface with a slave device having an access time of up to approximately 332ns and incur no wait states. This being the case, DRAMs with an access time of up to approximately 165ns could be accessed with 0-wait states.

The 8MHz PC-AT could interface with a slave device having an access time of up to approximately 250ns and incur no wait states. DRAMs with an access time of up to approximately 125ns could therefore be accessed with 0-wait states.

Transfers With 8-bit Devices

The ISA bus cycle types utilized to communicate with 8-bit devices include:

- Standard 8-bit device ISA bus cycle – four wait states
- Shortened 8-bit device bus cycle – one, two, or three wait states
- Stretched 8-bit device bus cycle – more than four wait states

The steps that follow describe the sequence of events that take place during an 8-bit bus cycle using the default READY# timing and explains how the default timing can be either shortened or stretched. Figure 6-1 illustrates an example bus cycle. The step numbers in the text that follows corresponds to the numbered reference points in figure 6-1.

1. The address being presented by the current bus master begins to appear on the LA bus at the start of the address phase. This corresponds to the leading edge of Ts. If the system board is based on an 80286 or 80386 microprocessor and address pipelining is asserted, the address may actually be present on the LA bus prior to the beginning of the bus cycle (as is the case in this example). 16-bit ISA memory expansion cards can use the portion of the address on the LA bus to perform an early address decode. 8-bit ISA expansion cards do not have access to the LA bus and therefore cannot perform an early address decode. I/O cards only use the lower 16 address bits and therefore cannot take advantage of address pipelining.
2. BALE is asserted half-way through the address phase, gating the address through the system board address latch onto the SA bus.
3. If this is a write bus cycle, the microprocessor's write data is gated onto the SD bus half-way through the address phase. It remains on the SD bus until half a BCLK cycle into the next bus cycle (half-way through the address phase of the next transfer).
4. The trailing-edge of BALE (at the beginning of the first data clock period) causes the system board address latch to latch the address being output by the CPU so that it remains static on the SA bus for the remainder of the bus cycle. The addressed slave device can safely complete the decoding process during this period.
5. If this is a memory transaction and the M16# signal is sampled deasserted by the system board bus control logic at the end of the address phase, the command line (SMRDC# or SMWTC#) is not activated until half-way through the first data clock period. If this is an I/O transaction, I/O16# will not be sampled until reference point seven to determine if the currently-addressed device is an 8 or a 16-bit device.
6. If this is a memory transaction and M16# was sampled deasserted at the end of the address phase, M16# is again sampled half-way through the first data clock period. The continued deasserted state of M16# indicates that the addressed expansion board is an 8-bit device.
7. The appropriate command line (SMRDC#, SMWTC#, IORC# or IOWC#) is asserted half-way through the first data clock period. During a transfer with an 8-bit device, the activation of the command line is delayed until the midpoint of the first data clock period to allow more time for address decode before command line activation. The command line then remains asserted until the end of the bus cycle (end of last Tc).
8. If this is an I/O transaction, the IO16# signal is sampled deasserted by the system board bus control logic, indicating that the addressed expansion board is an 8-bit device.
9. Half-way through the second data clock period and half-way through each subsequent data clock period, the default ready timer on the system board

samples the NOWS# line. If sampled asserted, the CPU READY# line is activated and the bus cycle ends on the next rising-edge of BCLK (the end of the current data clock period). In this way, an ISA board can terminate a bus cycle earlier than the default number of BCLK cycles (wait states) by activating NOWS#.

10. This item does not have a corresponding numbered reference point on the timing diagram. A bus cycle addressing an 8-bit ISA device defaults to six BCLK cycles (four wait states) if the following two conditions are met:
    a) the bus cycle isn't terminated earlier by the assertion of NOWS#.
    b) CHRDY is asserted when sampled during the first half of the last data clock period of the default cycle (first half of the 5th data clock period).

    This causes the duration of an ISA bus cycle when accessing an 8-bit device to default to four wait states (unless shortened by NOWS# to three, four, or five BCLK cycles, or lengthened by the deassertion of CHRDY. The bus cycle ends at the trailing-edge of the fifth data clock period.
11. During a read bus cycle, the microprocessor reads the data on the data bus at the trailing-edge of the last data clock period (Tc) of the bus cycle and the bus cycle is then terminated. The command line (SMRDC#, IORC#, etc.) is de-activated at that time. When a write bus cycle terminates, the MWTC#, SMWTC# or IOWC# command line is de-activated. Write data remains on the SD bus until half-way through the address phase of the next bus cycle. This accommodates the hold time of the device being written to and doesn't disturb the device being addressed in the next bus cycle because the command line for that bus cycle hasn't been activated yet.

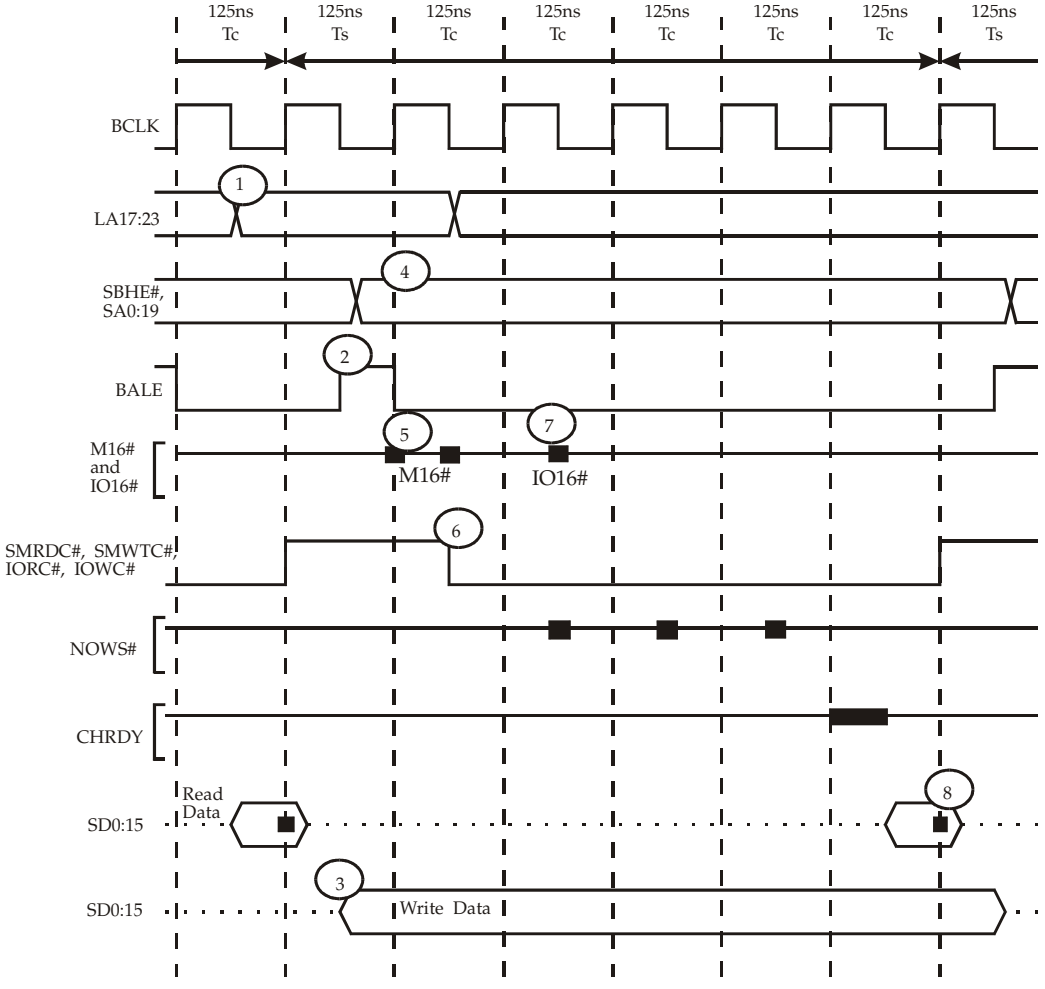


Figure 6-1. Standard Access to an 8-bit ISA Device

Transfers With 16-bit Devices

The ISA bus cycle types utilized to communicate with 16-bit devices include:

- Standard 16-bit device ISA bus cycle (Memory & I/O) — one wait state
- Shortened 16-bit device ISA bus cycle (Memory only) — zero wait states
- Stretched 16-bit device ISA bus cycle — more than one wait state

Standard 16-bit Memory ISA bus Cycle

Figure 6-2 illustrates the timing of a bus cycle on the ISA bus when the current bus master is communicating with a one wait state 16-bit memory device. Each of the numbered steps corresponds to the numbered reference points in figure 6-2.

1. If the system board is based on an 80286 or 80386 microprocessor and address pipelining is asserted, the address is present on the LA bus prior to the beginning of the bus cycle. This allows the addressed memory slave to start decoding the address early which may speed up access.
2. BALE is asserted halfway through the address phase. On the rising-edge of BALE, 16-bit ISA memory devices can begin to decode the LA lines to determine if the address is for them. When BALE is asserted, the lower portion of the address from the processor (A[19:0]) is transferred through the system board's address latch onto SA[19:0].
3. The addressed memory board activates M16# as a result of decoding the LA lines, indicating to the system board's bus control logic that it is capable of handling a 16-bit transfer without data bus steering being performed by the steering logic on the system board.
4. If this is a write bus cycle, the microprocessor's output data is gated onto the SD bus half-way through the address phase and remains on the SD bus until half a BCLK cycle into the next bus cycle (half-way through the address phase of the next bus cycle).
5. At the end of the address phase, the trailing-edge of BALE causes two events to take place:
   a) 16-bit ISA memory devices latch the LA lines so the addressed device is not deselected when the LA lines are pipelined with the address for the next transaction before the end of the current bus cycle.
   b) the address latch on the system board latches the lower twenty bits of the address, SA[19:0], so that they remain static on the SA bus for the remainder of the bus cycle. Slave devices can safely decode the SA address on the bus on the falling edge of BALE (if they haven’t done so already).
6. The system board's bus control logic samples M16# at the end of the address phase to determine if the addressed device can take advantage of the MRDC# or MWTC# command lines being asserted immediately. The appropriate command line (MRDC# or MWTC#) is asserted at the leading-edge of the first data clock period if M16# is sampled asserted. This command line remains asserted until the end of the bus cycle (end of last Tc). If M16# is sampled deasserted, the command line (MRDC# or MWTC#) is activated half-way through the first data clock period.

7. If M16# wasn't sampled asserted at the end of the address phase, the system board's bus control logic samples M16# a second time at the midpoint of the first data clock period to determine if data bus steering is necessary. Since this is an access to a 16-bit device, M16# is sampled asserted and steering is therefore unnecessary. Also at the midpoint of the first data clock period, the default ready timer on the system board samples NOWS#. If sampled asserted, the microprocessor's READY# line is asserted and the bus cycle terminates at the end of the first data clock period. In this way, a 16-bit ISA memory board can complete a bus cycle in two BCLK cycles (it should be noted, however, that the default ready timer ignores NOWS# during I/O bus cycles).
8. During address pipelining, the microprocessor is free to output the address for the next bus cycle during the current bus cycle. Only the upper portion of the pipelined address appears on the LA bus at this time because these bits are buffered but not latched from the microprocessor's address bus. The remainder of the address doesn't appear on the SA bus until the midpoint of the address phase in the next bus cycle.
9. CHRDY is sampled by the default ready timer at the beginning of the second data clock period to determine if the device will be ready to complete the bus cycle at the end of this BCLK cycle. If the device cannot complete the bus cycle by the end of this BCLK cycle, it should deassert CHRDY. If CHRDY is sampled deasserted by the default ready timer, it responds by extending the bus cycle by adding another data clock period. CHRDY is then checked at the beginning of each additional data clock period until the device releases CHRDY, indicating that the bus cycle can be completed.
10. An ISA 16-bit memory bus cycle defaults to three BCLK cycles (one wait state) if the bus cycle isn't terminated earlier by the assertion of NOWS# and if CHRDY stays asserted throughout the bus cycle. This means that the length of an ISA bus cycle when accessing a 16-bit memory card defaults to one wait state unless shortened by NOWS# or lengthened by CHRDY. READY# is then asserted to the microprocessor, telling it to read the data from the data bus (if this is a read transaction). When a memory write bus cycle terminates, the MWTC# command line is desasserted, but the data remains on the SD bus during the first half of the address phase in the next bus cycle. This provides hold time for the device being written to and doesn't affect the device being addressed in the next bus cycle because the command line hasn't been activated yet.

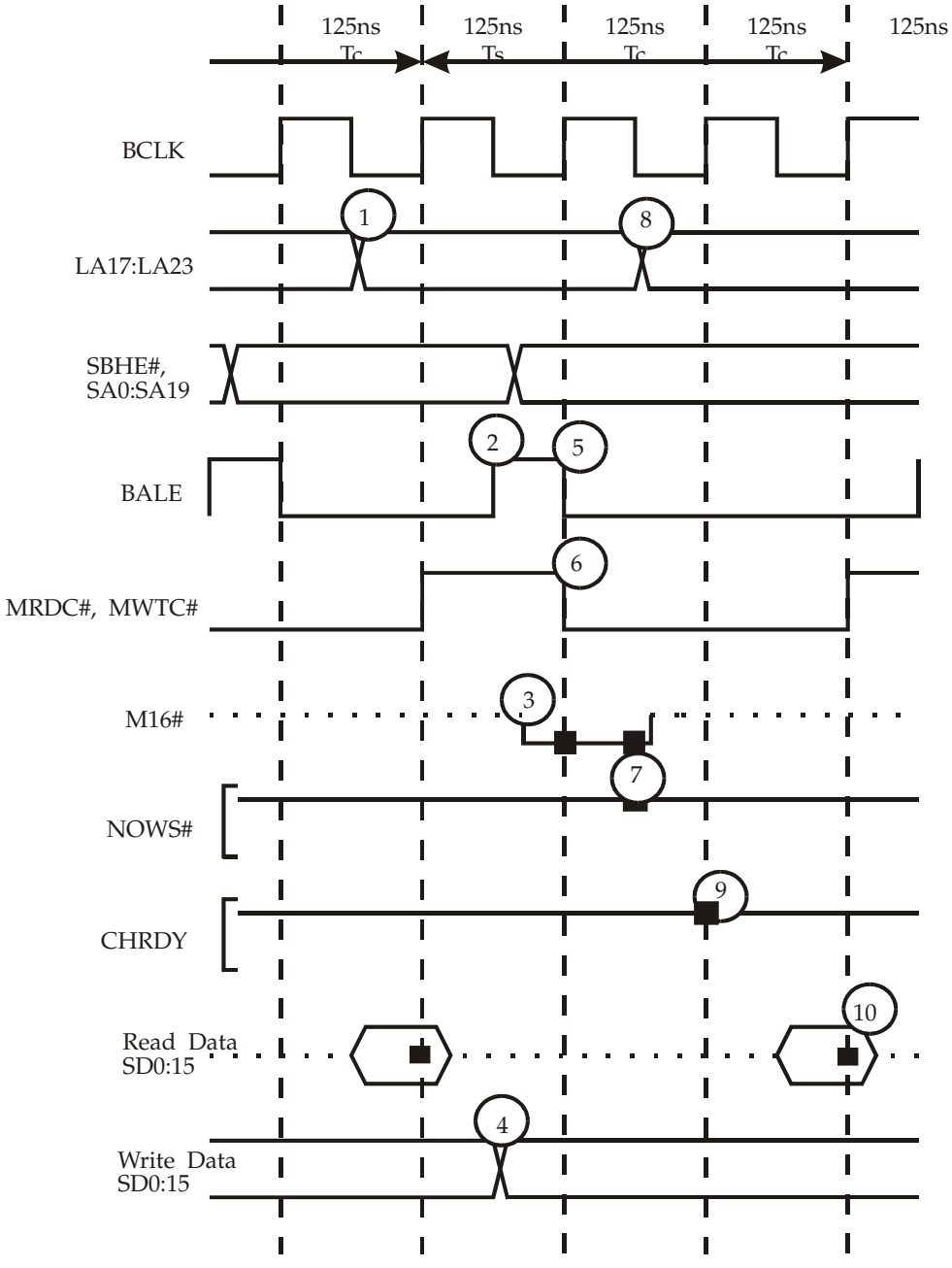


Figure 6-2. Standard Access to a 16-bit ISA Memory Device

Standard 16-bit I/O ISA bus Cycle

Figure 6-3 illustrates the timing of a bus cycle on the ISA bus when the current bus master is communicating with a 16-bit I/O device. Each of the numbered steps corresponds to the numbered reference points in figure 6-3.

1. If the system board is based on an 80286 or 80386 microprocessor and address pipelining is active, the address is present on the LA bus prior to the beginning of the bus cycle. The LA bus has no impact on I/O bus cycles since A[23:16] always contain zeros during I/O operations.
2. BALE is asserted halfway through the address phase, gating the address through the system board's address latch onto the SA bus.
3. If this is a write bus cycle, the microprocessor's output data is gated onto the SD bus half-way through the address phase and remains on the SD bus until half a BCLK cycle into the next bus cycle (half-way through the address phase of the next bus cycle).
4. At the start of the first data clock period, the trailing-edge of BALE causes the address latch on the system board to latch the lower twenty bits of the address, SA[19:0], so that it remains static on the SA bus for the remainder of the bus cycle. Slave devices can safely latch the SA address on the bus on the falling edge of BALE (if they haven't done so already).
5. The appropriate command line (IORC# or IOWC#) is also asserted at the midpoint of the first data clock period. This command line remains asserted until the end of the bus cycle (end of last Tc).
6. At the midpoint of the first data clock period, the default ready timer on the system board ignores the NOWS# line (since an I/O device is being accessed). This is done to prevent two back-to-back I/O write bus cycles from accessing the I/O device too quickly. This could violate the I/O write recovery time of the I/O device, causing improper operation.
7. During address pipelining, the microprocessor is free to output the address for the next bus cycle during the current bus cycle. Only the upper portion of the pipelined address appears at this time on the LA bus because these bits are buffered but not latched from the microprocessor's address bus. The remainder of the new address does not appear on the SA bus until the midpoint of the address phase of the next bus cycle.
8. IO16# is sampled at the midpoint of the second data clock period to determine if the I/O device is an 8 or 16-bit device. If sampled asserted, data bus steering is not performed and the bus cycle is terminated on the next rising-edge of BCLK (the end of the second data clock period). The bus cycle is not terminated if the CHRDY line is sampled deasserted.
9. CHRDY is sampled by the default ready timer at the end of the second data clock period to determine if the device is ready to complete the bus cycle. If

the device cannot complete the bus cycle by the end of the second BCLK cycle, it must deassert CHRDY. If CHRDY is sampled deasserted by the default ready timer, it responds by extending the bus cycle by one data clock period. CHRDY is then checked at the beginning of each additional data clock period until the device releases CHRDY to indicate that the bus cycle can be completed.

10. An ISA 16-bit I/O bus cycle defaults to three BCLK cycles (one wait state) if CHRDY stays asserted throughout the bus cycle. The bus cycle cannot be terminated earlier by the assertion of NOWS#. This means that the length of an ISA bus cycle when accessing a 16-bit I/O card defaults to one wait state unless lengthened by the deassertion of CHRDY. READY# is then asserted to the microprocessor, telling it to read the data from the data bus (if this is a read transaction). When an I/O write bus cycle terminates, the IOWC# command line is de-activated, but the data remains on the SD bus until the end of the first half of the address phase in the next bus cycle. This accommodates the hold time of the device being written to and doesn't disturb the device being addressed in the next bus cycle because the command line for that bus cycle hasn't been activated yet.

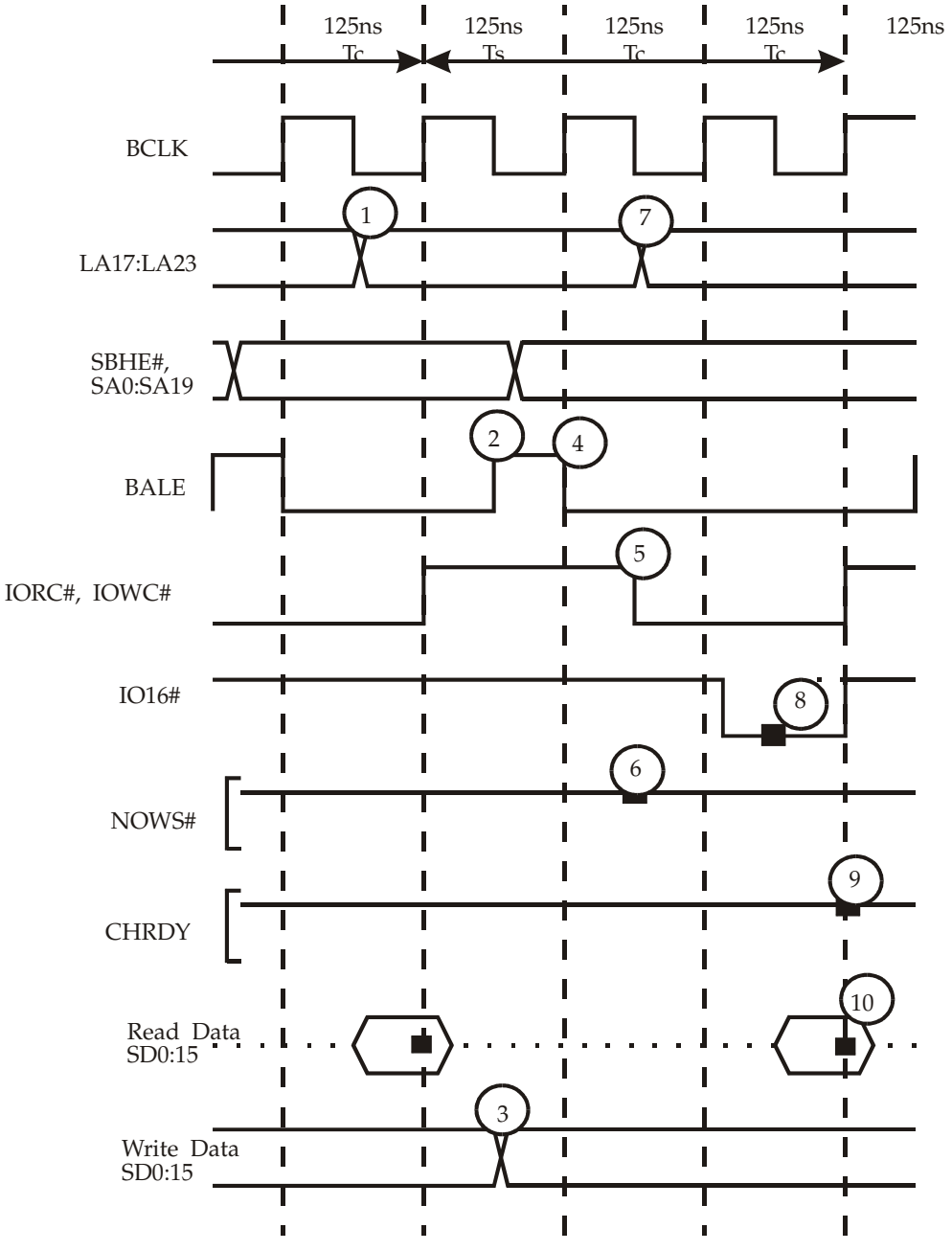


Figure 6-3. Standard Access to 16-bit I/O Device

Zero Wait State ISA Bus Cycle Accessing 16-bit Device

Figure 6-4 illustrates the timing of a bus cycle on the ISA bus when the current bus master is communicating with a zero wait state 16-bit device. Note that only 16-bit memory devices can complete bus cycles at zero wait states. Each of the numbered steps corresponds to the numbered reference points in figure 6-4.

1. In this example, address pipelining is active and the LA address is present on the ISA bus prior to the beginning of the bus cycle. This allows the addressed slave to start decoding the address early. In some cases, this allows a device to operate at zero wait states.
2. BALE is asserted halfway through the address phase. On the rising-edge of BALE, 16-bit ISA memory devices can begin to decode the LA lines to determine if the address is for them. When BALE is asserted, the lower portion of the address from the processor is transferred through the system board's address latch onto the SA bus.
3. The addressed memory board asserts M16# as a result of decoding the LA lines, indicating to the system board's bus control logic that it is capable of handling a 16-bit transfer without data bus steering being performed by the steering logic on the system board.
4. If this is a write bus cycle, the microprocessor's output data is gated onto the SD bus half-way through the address phase and remains on the SD bus until half a BCLK cycle into the next bus cycle (half-way through the address phase of the next bus cycle).
5. At the end of the address phase, the trailing-edge of BALE causes two events to take place:
   a) 16-bit ISA memory devices latch the LA lines so the addressed device will not be deselected if the LA lines are pipelined before the end of the current bus cycle.
   b) the address latch on the system board latches the lower twenty bits of the address, SA[19:0], so that they remain static on the SA bus for the remainder of the bus cycle. Slave devices can safely decode the SA address on the bus on the falling edge of BALE (if they haven't done so already).
6. The system board's bus control logic samples M16# at the end of the address phase to determine if the addressed device can take advantage of the MRDC# or MWTC# command line being asserted immediately. M16# is sampled asserted and the appropriate command line (MRDC# or MWTC#) is asserted at the leading edge of the first data clock period. This command line remains asserted until the end of the bus cycle (end of Tc).

7. If M16# was not sampled asserted the firts time, the system board's bus control logic samples M16# a second time at the midpoint of the first data clock period to determine if data bus steering is necessary. Since this is an access to a 16-bit device, no steering is necessary. Since the M16# line is asserted, the default ready timer samples the NOWS# line to determine if the bus cycle can end in zero wait states. In this example, M16# is sampled asserted, forcing the default ready timer to assert the microprocessor's READY# line before the end of the current data clock period. In this way, faster ISA memory boards can complete a bus cycle in two rather than three BCLK cycles.
8. Since the LA lines have already done their job, (the addressed device has already decoded the LA lines and latched the chip select), the microprocessor is free to output the address for the next bus cycle.
9. During a read, the microprocessor latches the contents of the data bus, thereby ending the bus cycle. During a write, the microprocessor ends the bus cycle.

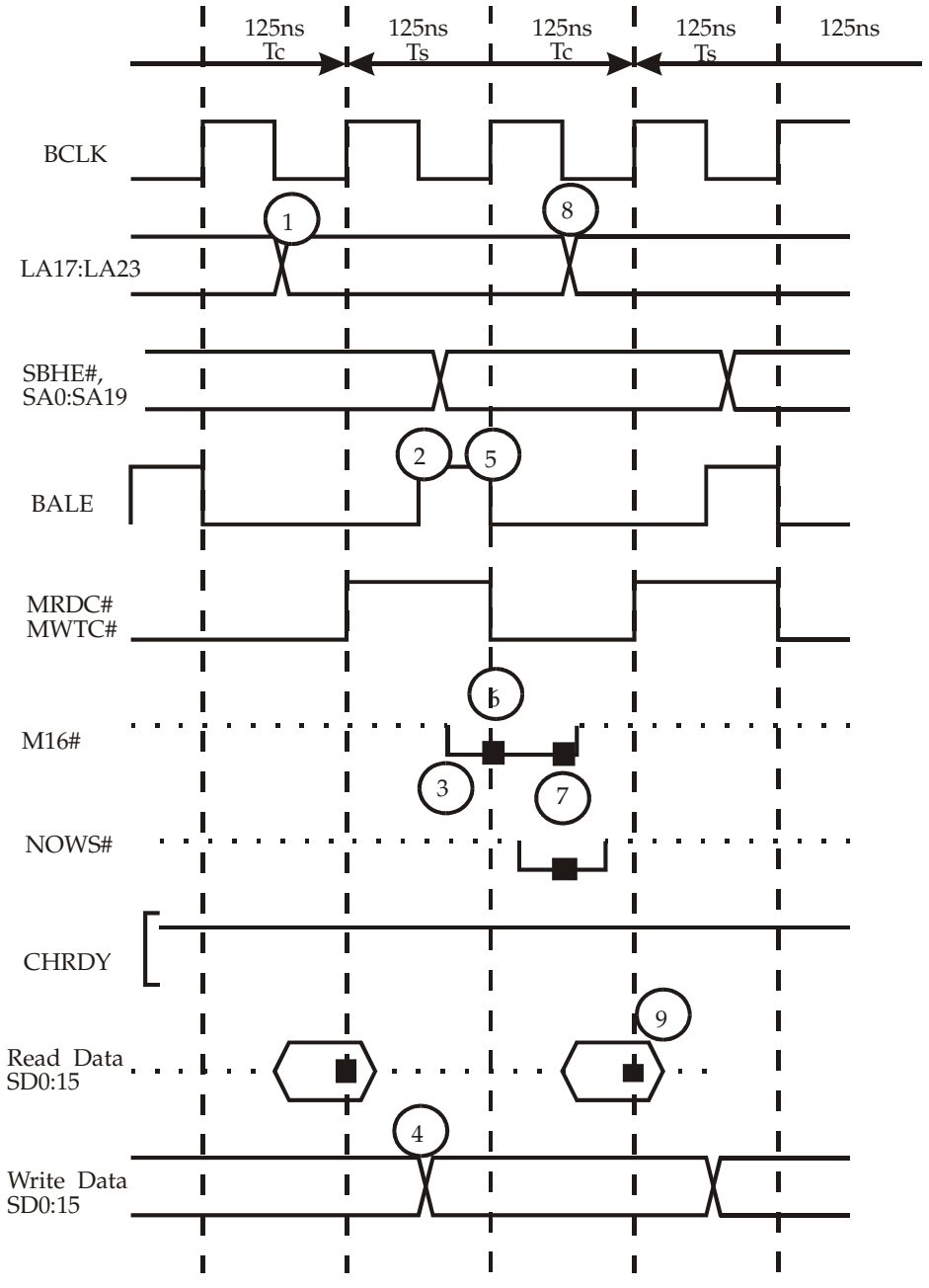


Figure 6-4. Zero Wait State Access to a 16-bit ISA Memory Device

ISA DMA Bus Cycles

ISA DMA Introduction

ISA machines use two Intel 8237 DMACs on the system board to implement the DMA logic. One of the DMACs is connected to the other in a master/slave configuration using channel zero on the master as the cascade input from the slave. Since each 8237 DMAC provides four DMA channels and one on the master is used as the cascade input from the slave, the ISA system provides a total of seven DMA channels. The four inputs to the slave DMAC are designated as channels zero – three, while the three inputs to the master are designated as channels five – seven.

In addition, the ISA machine implements the two DMACs in such a fashion that the three channels on the master (channels five – seven) are capable of performing 16-bit transfers, while the four channels on the slave (channels zero – three) are capable of performing 8-bit transfers.

Each DMA block data transfer can consist of up to 64K individual transfers. This limitation is imposed by the 16-bit transfer count register associated with each channel. This allows each of the channels to transfer up to 64KB of data. The 16-bit channels are also restricted to 64KB transfers because the DMA logic cannot increment the memory address across a 64K address boundary.

Each DMA channel can address any memory location within the 16MB range from 000000h to FFFFFFh. This limitation is imposed by the combination of the 16-bit memory address register associated with each channel in the DMAC and the 8-bit page register associated with each channel. This pair of registers associated with each channel provide a 24-bit memory address capability.

ISA expansion boards can become bus masters if they are connected to one of the three 16-bit DMA channels on the master DMAC (channels five – seven) and if the channel is programmed as a cascade channel. The ISA board may then request the use of the bus through the auspices of the master DMAC on the system board.

When a DMAC is bus master, it uses its own clock when executing bus cycles. This clock is referred to as the DMA clock and is 1/2 the BCLK frequency. Depending on the system design and the selected processor speed, this will yield a DMA clock of either 3MHz (6MHz AT), 4MHz (8MHz ISA-compatible machine), or 4.165MHz (8.33MHz ISA-compatible machine).

8237 DMAC Bus Cycle

The 8237 is built around a state machine with seven possible states, each one DMA clock period wide. Table 6-1 lists the clock period for the three possible processor speed settings.

Table 6-1. DMA Clock Speeds

Speed Setting	DMA Clock Frequency	DMA Clock Period
6MHz	3MHz	333.3ns
8MHz	4MHz	250ns
8.33MHz	4.165MHz	240ns

Prior to receiving a DMA Request, the DMAC is in the idle state (Si). When a DREQ is sensed, the DMAC enters a state where it asserts HOLD (Hold Request) to the microprocessor and awaits the HLDA (Hold Acknowledge). This state is called SO (the letter O). The DMAC remains in the SO state until HLDA is sensed asserted.

The DMAC can then proceed with the DMA transfer. S1, S2, S3 and S4 are the states used to execute a transfer (of a byte or word) between the requesting I/O device and system memory. In addition, when accessing a device that is slow to respond, a DMA transfer cycle can be stretched by de-asserting the DMAC's READY input until the device is ready to complete the transfer. This causes the DMAC to insert wait states (Sw), in the bus cycle until READY is asserted again.

The actions described in table 6-2 take place during states S1 – S4.

Table 6-2. ISA DMA State Table

State	Actions Taken
S1	During block and demand transfers, the middle byte of the memory address, A[15:8], only changes once every 256th transfer. For this reason, the DMAC only enters the S1 state every 256th transfer in order to update the middle byte of the address that is contained in the external DMA address latch. Starting at the trailing-edge of S1, the middle byte of the memory address is output onto data bus pins D[7:0] and is then latched into the external DMA address latch during S2. The DMAC also asserts AEN, causing the external DMA address latch to output and acting as an enable for the DMA page register addressing.
S2	During S2, the lower byte (A[7:0]) of the memory address is output directly onto address bus signals A[7:0]. If S2 was preceded by S1, the DMAC pulses its ADSTB output, causing the new middle byte of the address to be latched into the external DMA address latch. If S2 wasn't preceded by S1, ADSTB isn't pulsed, but the DMAC's AEN output is asserted. This causes the external DMA address latch to output the previously latched middle byte and acting as an enable for the DMA page register addressing. In addition, DAKn# is asserted to tell the I/O device that the transfer is in progress.
S3	S3 only occurs in a bus cycle if compressed timing hasn't been selected for this DMA channel. See text below for a discussion of compressed timing. During S3, the MRDC# or the IORC# line is asserted. If the DMA channel is programmed for extended writes, the MWTC# or IOWC# line is also kept asserted during S3.
S4	If the DMA channel was not programmed for extended write, the MWTC# or IOWC# is asserted at the start of S4. If extended write is selected, the write command line was already asserted at the start of S3. The actual read/write takes place at the trailing-edge of S4 when both the read and write command lines are de-asserted by the DMAC. This completes the transfer of a byte or word between memory and the requesting I/O device.

When compressed timing is selected, S3 is eliminated from the DMA transfer cycle. The only real purpose of S3 is to allow the read command line to be asserted for twice the duration that it is when compressed timing is active. Not all memory and I/O devices will tolerate this abbreviated read command duration, so it must be used cautiously.

When extended write is selected, it causes the write command line to be asserted during S3 rather than S4, effectively doubling the duration of the write command line's assertion period.

It should be obvious that extended write and compressed timing are mutually exclusive because S3 is essential for extended write and is eliminated when compressed timing is selected.

Table 6-3 illustrates the transfer speeds possible at the three clock speeds under the following conditions:

- Compressed timing turned off
- Compressed timing turned on

The table assumes that the transfer is no more than 256 bytes in length. This was assumed for simplicity's sake. Every 256 transfers the DMAC must insert an S1 state in the next bus cycle to update the middle byte of the memory address (A[15:8]), which must be latched into the external DMA address latch. This adds one DMA clock period to the duartion of every 257th bus cycle.

Table 6-3. ISA DMA Transfer Rates

	Compressed Off			Compressed On		
DMA Clock Frequency	3MHz	4MHz	4.165MHz	3MHz	4MHz	4.165MHz
Transfers per Second	1M/s	1.3M/s	1.39M/s	1.5M/s	2M/s	2.08M/s

When looking at table 6-3, keep in mind that each bus cycle consists of three DMA clock cycles with compressed timing turned off and two DMA clock cycles with compressed timing turned on.

Chapter 7

The Previous Chapter

The previous chapter provided a review of bus master and DMA bus cycles in the ISA environment.

This Chapter

This chapter provides a detailed description of the EISA CPU and bus master bus cycle types.

The Next Chapter

The next chapter provides a detailed description of the EISA DMA bus cycle types.

Intro to EISA CPU and Bus Master Bus Cycles

In order to maintain complete ISA compatibility, ISA bus cycles are executed precisely as they are in an ISA machine. These bus cycle types have been described in the preceding chapter.

As stated earlier, an Intel x86-compatible processor is capable of executing seven types of bus cycles:

- Memory data read and memory instruction read. These two types are actually identical, reducing the total to six bus cycle types.
- Memory data write
- I/O data read
- I/O data write
- Interrupt acknowledge
- Halt or Shutdown (also referred to as the special cycle)

Of these six, only four are ever seen by the expansion boards on the ISA bus:

- Memory Read

- Memory Write
- I/O Read
- I/O Write

In an EISA system, the main CPU is capable of performing three variants of each of these four bus cycle types when communicating with a device over the EISA bus:

- Standard timing
- Compressed timing (not implemented in current machines)
- Burst timing

EISA bus masters are capable of executing two of these three variants:

- Standard timing
- Burst timing

Standard EISA Bus Cycle

General

The standard EISA bus cycle type is based upon a zero wait state bus cycle. Unless wait states are inserted by the slave, the transaction completes in two BCLK periods. Each wait state adds one additional BCLK period. The following formula is used to calculate the total transfer time:

Total Transfer Time = N * (2+Tw) * (1 BCLK period)

where: Tw = number of wait states per bus cycle
N = number of bus cycles for overall transfer

As an example, a transfer of 64 doublewords (256 bytes) completes in 15.36 microseconds for a 32-bit transfer with a 8.33MHz BCLK, while a 16-bit transfer completes in 30.72 microseconds. This example assumes that no preempts occur during the transfer and the addressed slave is a zero wait state device.

Analysis of EISA Standard Bus Cycle

The timing diagram in figure 7-1 illustrates the timing for three bus cycles, the first of which has one wait state and the last two are zero wait state bus cycles. The numbered steps that follow correspond to the reference points in the illustration.

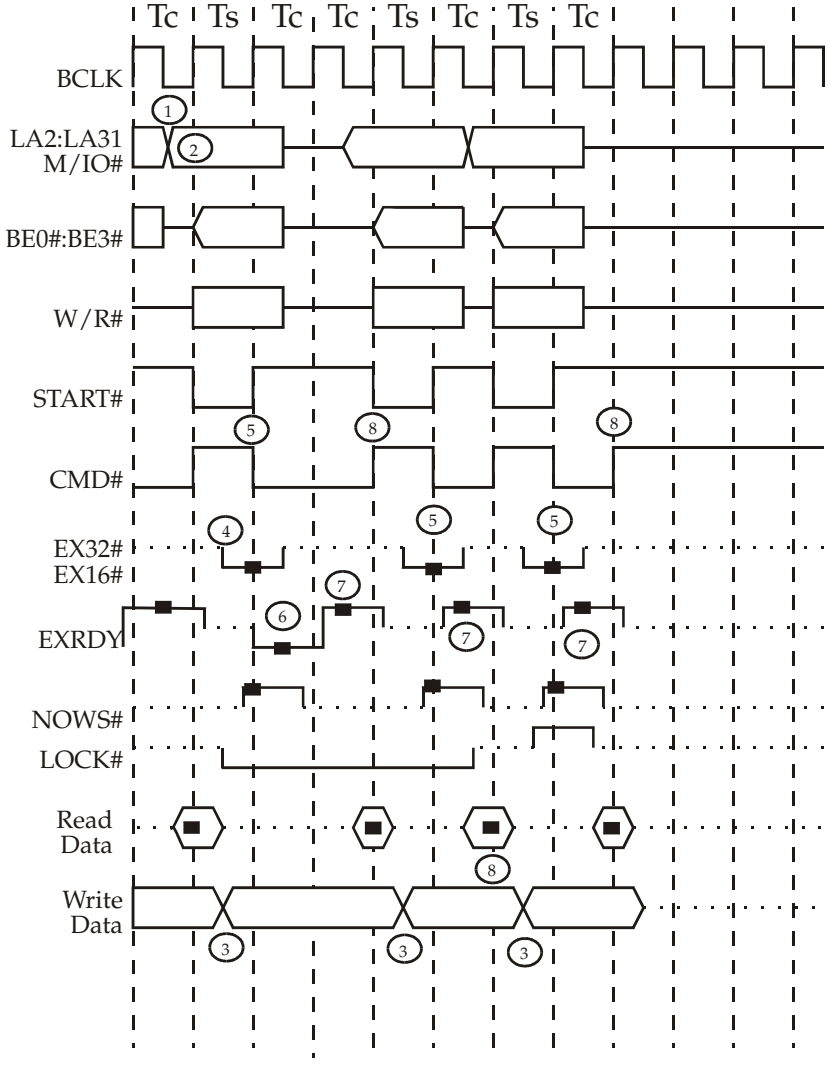


Figure 7-1. The EISA Standard Bus Cycle

1. The first bus cycle after bus grant cannot use address pipelining. After the first bus cycle, however, the bus master can use address pipelining to output the address and M/IO# early.
2. After the bus master (or CPU) has requested and been granted the bus, the bus cycle begins on the rising edge of BCLK (the leading-edge of Ts) with the assertion of the START# signal by the current bus master. START# remains asserted for a full BCLK cycle (all of Ts). At the leading-edge of START#, the bus master or CPU places the address on the LA bus and byte enables and also outputs M/IO#. If address pipelining is active, the address, byte enables and M/IO# may be placed on the bus during the previous bus cycle. W/R# is set to the appropriate state at the beginning of the bus cycle.
3. If a write bus cycle is in progress, the bus master begins to drive the data onto the appropriate data paths at the midpoint of Ts.
4. The addressed EISA slave decodes the address and asserts either EX16# or EX32# indicating that it is an EISA device and the data size it's prepared to handle. I/O devices should also ensure that the AEN signal is deasserted before decoding an address. AEN is asserted by the DMA controller when it is placing a valid memory address on the address bus. In order to maintain ISA bus master compatibility, an EISA I/O slave should assert IO16# as well as EX16# or EX32#. EISA slaves that do not need to maintain ISA bus master compatibility do not need to assert IO16#. The system board develops M16# from EX16# or EX32# to maintain ISA bus master compatibility when communicating with ISA memory slaves. Note that EISA compressed mode is not supported in current implementations of EISA; however, if implemented the addressed slave should assert NOWS# prior to the end of Ts.
5. If the addressed slave must latch the address information, it should be latched on the trailing-edge of START#. The system board's data bus steering logic samples the EX16# and EX32# lines to determine if steering is necessary. CMD# is asserted by the system board coincidentally with the deassertion of START# by the bus master. Only the system board drives the CMD# line. CMD# then remains asserted until the end of the bus cycle. If support for EISA compressed bus cycles were implemented, the main CPU logic would sample NOWS# at the trailing-edge of start to determine if the addressed slave supports EISA compressed mode bus cycles.
6. EXRDY is sampled at the falling edge of every BCLK after CMD# is asserted. If sampled deasserted, the bus cycle is extended by one wait state (an additional Tc). Designers of EISA expansion cards are guaranteed that the address presented on the LA bus, the byte enable lines and the state of M/IO# will remain static until the midpoint of the first Tc period of the bus cycle.

7. If EXRDY is sampled asserted at the midpoint of Tc, the bus cycle is terminated at the end of Tc. If the current bus master has another bus cycle to perform and it uses address pipelining, the address for the next bus cycle is placed on the LA bus, the byte enable lines and M/IO#.
8. After EXRDY is sampled asserted at the midpoint of Tc, the bus cycle is terminated at the end of the BCLK cycle. The system board logic deasserts the CMD# signal. If a read bus cycle is in progress, the bus master reads the data from the data bus. If a write bus cycle is in progress, the bus master ends the bus cycle but continues to drive the data onto the data bus until the midpoint of Ts of the next bus cycle. This is done to ensure that the hold time for the currently-addressed device is satisfied.

Performance Using EISA Standard Bus Cycle

Assuming that the current bus master and the currently-addressed slave are both 32-bit devices, the BCLK frequency is 8.33MHz, and the bus master performs a series of 32-bit transfers, the transfer rate would be 16.66MB/second:

120ns per BCLK cycle x 2 BCLK cycles per transfer
= 240ns per transfer, divided into one second
= 4.166M transfers/second, at 4 bytes/transfer
= 16.66MB/second

If the currently-addressed slave is a 16-bit device, the transfer rate would be 8.33MB/second.

Compressed Bus Cycle

General

To the authors' knowledge, currently-available EISA chipsets do not support the EISA compressed bus cycle. For this reason, a detailed analysis of the compressed bus cycle is reserved for a future printing.

Only the main CPU can utilize EISA compressed bus cycles when communicating with EISA memory or I/O slaves that support compressed mode. Using the compressed bus cycle, the CPU can complete a transfer every 1.5 BCLK cycles. The following formula may be used to calculate the overall transfer rate when transferring a block of data between the main CPU and a slave that supports compressed bus cycles:

Total Transfer = N * (1.5 BCLK periods)

where: N = the total number of bus cycles for the overall block transfer

As an example, a transfer of 64 doublewords (256 bytes) completes in 11.52 microseconds for a 32-bit transfer with a 8.33MHz BCLK, while a 16-bit transfer completes in 23.04 microseconds. This example assumes that no preempts occur during the transfer and that the addressed slave is a zero wait state device.

Using the compressed bus cycle, the CPU presents a new address every 1.5 BCLK periods (instead of two) and the system board shortens the duration of the CMD# assertion period to one-half of a BCLK period.

If a slave supports compressed bus cycles, it must assert NOWS# prior to the end of Ts. The slave must not de-assert EXRDY after asserting NOWS#. If the system board samples NOWS# asserted at the leading-edge of CMD# and the system board design supports compressed mode, the CMD# pulse width is shortened to .5 BCLK periods. Since the main CPU logic might not support compressed mode, or the current bus master might not be the main CPU, the slave must be prepared to accept CMD# with a duration of one BCLK or longer.

Performance Using Compressed Bus Cycle

If both the main CPU and the currently-addressed slave support compressed mode, the BCLK frequency is 8.33MHz, and both the master and the salve are 32-bit devices, the transfer rate for a block data transfer would be 22.22MB/second:

120ns per BCLK cycle x 1.5 BCLK cycles per transfer
= 180ns per transfer, divided into one second
= 5.55M transfers/second, at 4 bytes/transfer
= 22.22MB/second

If the currently addressed slave is a 16-bit device, the transfer rate would be 11.11MB/second.

Burst Bus Cycle

General

A burst transfer is used to transfer blocks of data between the current bus master and EISA memory. A burst must consist of all reads or all writes. Reads and writes may not be mixed within a burst. In other words, the state of the W/R# bus cycle definition line may not be changed during a burst. After the initial transfer in a block data transfer, each subsequent EISA Burst bus transfer can be completed in one BCLK period. The initial transfer requires the time periods consisting of Ts and Tc to transfer the first data item and for the master and slave to agree to use burst mode for the subsequent transfers. Unless wait states are inserted by the slave, each subsequent transfer can then be completed in one BCLK period. Each wait state adds one additional BCLK period. The following formula is used to calculate the total transfer time:

Total Transfer Time = (1 + Twi + N) * one BCLK period

where: Twi = wait states inserted per transfer
N = number of bus cycles for overall transfer

As an example, a transfer of 64 doublewords (256 bytes) completes in 7.8 microseconds for a 32-bit transfer with a 8.33MHz BCLK, while a 16-bit transfer completes in 15.6 microseconds. This example assumes that no preempts occur during the transfer and the addressed slave is a zero wait state device.

Analysis of EISA Burst Transfer

The timing diagram in figure 7-2 illustrates the timing for five transfers performed using burst mode. The following numbered steps correspond to the reference points in the illustration.

A 16-bit burst transfer is identical with the exception that EX16# is generated by the slave instead of EX32#.

1. The current bus master can use address pipelining to output the first address and M/IO# early.
2. At the beginning of the first bus cycle in the transfer, the current EISA bus master activates the START# signal. Assertion of START# indicates that the bus master has placed a valid address and bus cycle definition on the bus. The EISA bus controller (EBC) on the system board samples START# asserted and recognizes that an EISA bus master, rather than an ISA bus master, has initiated a bus cycle. In response, the EBC generates BALE during

Ts. This is done in case the EISA bus master is addressing an ISA device. In addition, the bus master sets the byte enable lines and W/R# to the appropriate state. W/R# remains in the selected state (write or read) throughout the burst transfer.

3. If this is a write transfer, the bus master starts to drive the data onto the data bus at the midpoint of Ts.
4. At the end of Ts, the current bus master and the system board logic sample EX16# and EX32#. The assertion of either of these signals indicates that the currently-addressed device is an EISA device and what data paths it is capable of using. The bus master deasserts START# and the system board logic asserts CMD# to indicate that the data phase has begun. If the bus master is capable of using burst transfers, it samples SLBURST# to determine if the addressed slave also supports burst. In this example, SLBURST# is sampled asserted, indicating that the slave supports burst mode.
5. In response to sampling SLBURST# asserted, the bus master asserts MSBURST# at the midpoint of Tc to indicate to the slave that it also supports burst mode and will use it for the remaining transfers in the burst. Also, the bus master samples EXRDY at the midpoint of Tc to determine if the addressed slave will be ready to complete the first transfer at the end of the current Tc. In this example, EXRDY is sampled asserted, indicating that the first transfer can be completed at the end of this Tc period. In response. the bus master pipelines out the second address starting at the midpoint of Tc.
6. At the end of the first Tc period, the bus master completes the first transfer in the burst. If a read burst is in progress, the bus master reads the data from the appropriate data paths. If a write burst is in progress, the bus master starts to drive the data for the second transfer onto the appropriate data paths. The slave samples MSBURST# at the end of each Tc period to determine if the bus master will use burst mode for the remaining transfers. In this example, MSBURST# is sample asserted, so the burst transfer continues.
7. At the midpoint of the second Tc, the bus master samples EXRDY to determine if the slave will be ready to complete the second transfer at the end of this Tc period. In this example, it is sampled asserted, indicating that the slave will be ready. In response, the bus master begins to drive the third address out at the midpoint of Tc.
8. At the end of the second Tc, the slave samples MSBURST# again to determine if the bus master is still bursting. The asserted state indicates that it is. The bus master completes the second transfer. If a read burst is in progress, the bus master reads the data from the appropriate data paths. If a write burst is in progress, the bus master starts to drive the data for the third transfer onto the appropriate data paths.

9. At the midpoint of the third Tc, the bus master samples EXRDY to determine if the slave will be ready to complete the third transfer at the end of this Tc period. In this example, EXRDY is sampled deasserted, indicating it will not be ready. This causes the bus master to insert a wait state of one Tc duration to stretch the data transfer time for the third transfer. If a read transfer is in progress, the bus master doesn't read the third transfer's data from the bus at the end of this Tc. If a write transfer, the bus master continues to drive the data for the third transfer onto the data bus during the next Tc. The bus master pipelines out the address for the fourth transfer, however, starting at the midpoint of the third Tc period.
10. At the midpoint of the fourth Tc, the bus master samples EXRDY to determine if the slave will be ready to complete the third transfer at the end of this Tc period. Since EXRDY is sampled asserted, it will be ready. The bus master does not pipeline out the address for the fifth transfer yet and continues to drive the data for the third transfer onto the data bus.
11. The bus master completes the third transfer. If a read burst is in progress, the bus master reads the data from the appropriate data paths. If a write burst is in progress, the bus master starts to drive the data for the fourth transfer onto the appropriate data paths.
12. At the midpoint of the fifth Tc period, the bus master samples EXRDY# to determine if the slave will be ready to end the fourth transfer at the end of the current Tc period. Since EXRDY is sampled asserted, the slave will be ready to end the transfer. The bus master also pipelines out the fifth address at the midpoint of Tc.
13. The bus master completes the fourth transfer. If a read burst is in progress, the bus master reads the data from the appropriate data paths. If a write burst is in progress, the bus master starts to drive the data for the fifth transfer onto the appropriate data paths.
14. At the midpoint of the sixth Tc, the bus master samples EXRDY# to determine if the slave will be ready to end the fifth transfer at the end of the current Tc period. Since EXRDY is sampled asserted, the slave will be ready to end the transfer. Since this is the end of the sample burst, the bus master de-activates MSBURST# to inform the slave that the last transfer of the burst is in progress. In this example, the bus master pipelines out the next address at the midpoint of Tc. In this example, the bus master is addressing a device other than the memory slave, causing the slave to release SLBURST#.
15. At the end of the sixth Tc period, the bus master completes the last transfer of the burst. If a read burst is in progress, the bus master reads the data from the appropriate data paths. If a write burst is in progress, the bus master ends the transfer and ceases to drive the data bus. This completes the example burst transfer.

16. The bus cycle following the burst is a standard EISA bus cycle. Since the bus master is setting W/R# low, it is a read. The bus master samples EXRDY asserted at the midpoint of Tc and reads the data from the data bus at the end of Tc and ends the bus cycle.
17. The next bus cycle is also a standard EISA bus cycle. The high on W/R# indicates that a write is in progress. The bus master begins to drive the data onto the data bus at the midpoint of Ts, samples EXRDY asserted at the midpoint of Tc, and ends the bus cycle at the end of Tc.

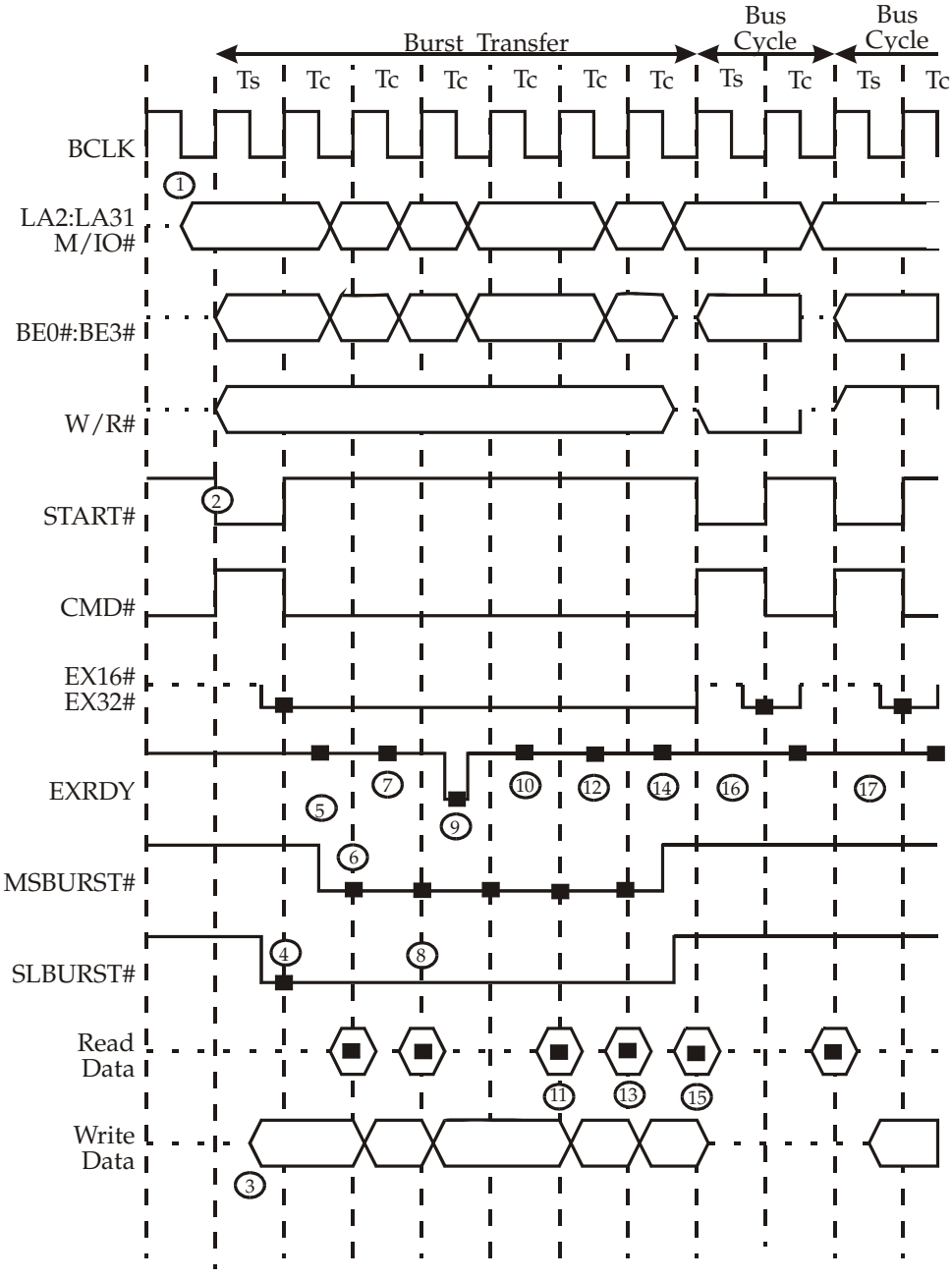


Figure 7-2. The EISA Burst Transfer

Performance Using Burst Transfers

Once a 32-bit bus master and a 32-bit slave have switched into burst mode, the second through the last transfers may be completed at the following rate:

8.33MHz BCLK = 120ns per BCLK cycle
1 second/120ns per transfer = 8.33M transfers/second
8.33M transfers/second x 4 bytes per transfer = 33.33MB/second

If the bus master and/or the slave are 16-bit devices, the maximum transfer rate would be 16.66MB/second.

DRAM Memory Burst Transfers

The addresses output by the bus master when bursting to or from Page Mode or Static Column (SCRAM) memory must be within a 1024 byte DRAM memory row (address lines LA[31:10] cannot change during the burst). The addresses within the burst do not have to be sequential. They only have to be within the same row. To change DRAM rows, the burst transfer must be terminated by the bus master by setting MSBURST# deasserted on the last cycle in the row, and the burst sequence is then restarted within a new row.

Downshift Burst Bus Master

A downshift burst bus master is a 32-bit burst bus master that can convert to a 16-bit burst bus master on-the-fly. In other words, if the bus master samples EX16# and SLBURST# asserted at the end of Ts, it automatically adjusts itself to only use the lower two data paths during the burst. The bus master is responsible for copying data to the appropriate data paths during the burst. The system board data bus steering logic will not take care of data copying. At the start of the first transfer in the burst, the downshift bus master must indicate its ability to downshift by setting MASTER16# asserted while START# is asserted (in other words, for the duration of the address phase).

Chapter 8

The Previous Chapter

The previous chapter described the bus cycle types that may be run by the main CPU or an EISA bus master.

This Chapter

This chapter describes the EISA DMA capability. This includes a description of the EISA DMA bus cycle types and other capabilities of the EISA DMA controller.

The Next Chapter

The next chapter provides an introduction to the bus structure hierarchy in a typical EISA system. It describes the distribution of functions between the host bus, EISA bus and the X-bus on the typical EISA system board and the relationship of the functional areas to each other.

DMA Bus Cycle Types

Introduction

The EISA DMA controller incorporates seven DMA channels, each capable of performing 8, 16 or 32-bit transfers. In addition, each DMA channel may be individually programmed to utilize one of four types of bus cycles when performing data transfers between an I/O device and memory. The following sections describe the bus cycle types and other DMA improvements. Detailed timing diagrams and register-level programming information may be found in the EISA specification.

Compatible DMA Bus Cycle

Description

Each of the seven DMA channels is default-programmed to use ISA-compatible DMA bus cycles to transfer data between an I/O device and memory. As in an ISA machine, channels zero – three are default-programmed for 8-bit transfers, while channels five – seven are default programmed for 16-bit transfers. Any DMA channel may be re-programmed to perform 8, 16, or 32-bit transfers using the ISA-compatible bus cycle.

When programmed to use ISA-compatible DMA bus cycles, a transfer is performed every eight BCLK periods. Table 8-1 defines the duration of key signals during an ISA-compatible DMA bus cycle.

Table 8-1. The DMA ISA-Compatible Bus Cycle

Event	Duration
Memory address present	8.0 BCLKs
Duration of data transfer period during a memory to IO transfer (CMD# active)	4.5 BCLKs
Duration of MRDC# during memory to IO transfer	4.5 BCLKs
Duration of IORC# during I/O to memory transfer	6.5 BCLKs
Duration of IOWC# during a memory to IO transfer	4.0 BCLKs
Duration of MWTC# during I/O to memory transfer	4.0 BCLKs

The duration of the key signals illustrated in table 8-1 defines the amount of time the memory and I/O device have to recognize that they are being addressed and to either accept or output data. Comparing this table to the tables found in the sections on the other three DMA bus cycle types, it is clear that the amount of time allotted for address decode and data movement becomes increasingly shorter for the faster bus cycle types.

Performance and Compatibility

Table 8-2 defines the data transfer rates when a DMA channel is programmed to use the ISA-compatible DMA bus cycle to transfer data.

Table 8-2. ISA-Compatible Transfer Rates

I/O Device Size	Transfer Rate
8-bit	1.0416MB/second
16-bit	2.0833MB/second
32-bit	4.1666MB/second

When programmed to use the ISA-compatible DMA bus cycle, a DMA channel may be used to transfer data between an ISA-compatible I/O device and memory.

Type A DMA Bus Cycle

Description

When programmed to use Type A DMA bus cycles, a transfer is performed every six BCLK periods. Table 8-3 defines the duration of key signals during a Type A DMA bus cycle.

Table 8-3. The DMA Type A Bus Cycle

Event	Duration
Memory address present	6.0 BCLKs
Duration of data transfer period during a memory to IO transfer (CMD# active)	3.5 BCLKs
Duration of IORC# during I/O to memory transfer	4.5 BCLKs
Duration of IOWC# during a memory to IO transfer	3.0 BCLKs

The duration of the key signals illustrated in table 8-3 defines the amount of time the memory and I/O device have to recognize that they are being addressed and to either accept or output data. When performing Type A bus cycles, the DMA controller uses W/R# rather than MRDC# or MWTC# to indicate the type of memory operation,.

Performance and Compatibility

Table 8-4 defines the data transfer rates when a DMA channel is programmed to use the Type A DMA bus cycle to transfer data.

Table 8-4. Type A Transfer Rates

I/O Device Size	Transfer Rate
8-bit	1.388MB/second
16-bit	2.777MB/second
32-bit	5.555MB/second

When a DMA channel is programmed to use the Type A DMA bus cycle to transfer data, the channel may be used to transfer data between fast, EISA memory and an I/O device designed for Type A transfers. In addition, many older, ISA I/O devices may also work with a channel programmed for Type A bus cycles. This is because the Type A transfer does not involve a significant amount of compression compared to the ISA-compatible bus cycle. Compatibility may be determined by testing.

Type B DMA Bus Cycle

Description

When programmed to use Type B DMA bus cycles, a transfer is performed every four BCLK periods. Table 8-5 defines the duration of key signals during a Type B DMA bus cycle.

Table 8-5. The DMA Type B Bus Cycle

Event	Duration
Memory address present	4.0 BCLKs
Duration of data transfer period during a memory to IO transfer (CMD# active)	2.5 BCLKs
Duration of IORC# during I/O to memory transfer	3.5 BCLKs
Duration of IOWC# during a memory to IO transfer	2.0 BCLKs

The duration of the key signals illustrated in table 8-5 defines the amount of time the memory and I/O device have to recognize that they are being addressed and to either accept or output data. When performing Type B bus cycles, the DMA controller uses W/R# rather than MRDC# or MWTC# to indicate the type of memory operation,.

Performance and Compatibility

Table 8-6 defines the data transfer rates when a DMA channel is programmed to use the Type B DMA bus cycle to transfer data.

Table 8-6. Type B Transfer Rates

I/O Device Size	Transfer Rate
8-bit	2.083MB/second
16-bit	4.166MB/second
32-bit	8.333MB/second

When a DMA channel is programmed to use the Type B DMA bus cycle to transfer data, the channel may be used to transfer data between fast, EISA memory and an I/O device designed for Type B transfers. In addition, some older, ISA I/O devices may also work with a channel programmed for Type B bus cycles. Although the Type B transfer involves a significant amount of compression compared to the ISA-compatible bus cycle, some ISA I/O devices may be fast enough to function correctly at this speed. Compatibility may be determined by testing.

Type C DMA Bus Cycle

Description

The Type C DMA bus cycle is very similar to the burst bus cycle run by a bursting EISA bus master or the main CPU. When the first bus cycle in a series is initiated, the DMA controller samples SLBURST# to determine if the addressed memory supports burst mode. In response to SLBURST# assertion, the controller then activates MSBURST# to indicate bursting will be used to transfer the data block. As with the other DMA bus cycle types, the controller uses the combination of DAKn# and either the IORC# or IOWC# line to address the I/O device. A byte, word or doubleword of data is transferred every BCLK cycle.

Performance and Compatibility

Table 8-7 defines the data transfer rates when a DMA channel is programmed to use the Type C DMA bus cycle to transfer data.

Table 8-7. Type C Transfer Rates

I/O Device Size	Transfer Rate
8-bit	8.33MB/second
16-bit	16.66MB/second
32-bit	33.33MB/second

When a DMA channel is programmed to use the Type C DMA bus cycle to transfer data, the channel may only be used to transfer data between fast, EISA memory and an I/O device designed for Type C transfers. No ISA I/O devices will work with a channel programmed for Type C bus cycles.

EISA DMA Transfer Rate Summary

Table 8-8 indicates the maximum data transfer rates achievable for each DMA bus cycle type, and the expansion devices that are compatible with the bus cycle type.

Table 8-8. EISA DMA Transfer Rates

Transfer Type	DMA Cycle Type	Transfer Rate (MB/sec)	Compatibility
ISA-compatible	8-bit	1.0	all ISA
	16-bit	2.0	all ISA
Type A	8-bit	1.3	most ISA
	16-bit	2.6	most ISA
	32-bit	5.3	EISA-only
Type B	8-bit	2.0	some ISA
	16-bit	4.0	some ISA
	32-bit	8.0	EISA-only
Type C (Burst)	8-bit	8.2	EISA-only
	16-bit	16.5	EISA-only
	32-bit	33.0	EISA-only

Other DMA Enhancements

Addressing Capability

The EISA DMA controller generates full 32-bit addresses, giving it the ability to transfer data to or from memory throughout the full 4GB address range.

Preemption

When a DMA channel is programmed for Type A, Type B, or Type C bus cycles, it may be preempted by the CAC if another device requires the use of the bus. When a channel is programmed for ISA-compatible DMA bus cycles, however, it cannot be preempted. This means that it can prevent other devices from receiving the use of the bus on a timely basis if the channel is programmed for a lengthy block or demand mode transfer. Care should therefore be exercised.

When the CAC detects another device that requires the use of the bus, it removes the bus grant from the DMA controller. The active DMA channel releases the bus within four microseconds.

Buffer Chaining

The EISA DMA controller's buffer chaining function permits the implementation of scatter write and gather read operations. A scatter write operation is one in which a contiguous block of data is read from an I/O device and is written to two or more areas of memory, or buffers. A gather read operation reads a stream of data from several blocks of memory, or buffers, and writes it to an I/O device.

The programmer writes the start address of the first memory buffer to the DMA channel and sets the channel's transfer count equal to the number of bytes, words, or doublewords to be transferred to or from the first buffer. The programmer then enables chaining mode, causing the DMA channel to load the start memory address and transfer count into another set of channel registers, known as the current registers. The programmer then writes the start address of the second memory buffer to the DMA channel and sets the channel's transfer count equal to the number of bytes, words, or doublewords to be transferred to or from the second buffer.

When the DMA channel has exhausted the first transfer count, the channel automatically loads the current registers from the secondary registers and generates either TC or an IRQ13. If the channel was programmed by the main CPU, IRQ13 is generated. If the channel was programmed by an EISA bus master, TC is generated instead. The TC or IRQ13 informs the bus master or microprocessor that the first buffer transfer has been completed, the second buffer transfer is in progress and the start address and transfer count for the third

buffer transfer (if there is one) should be written to the channel's registers. Updating these registers causes the controller to de-activate TC or IRQ13.

The channel generates a Transfer Complete (TC) when the transfer count is exhausted and the channel's registers have not been reloaded.

Ring Buffers

The EISA DMA controller allows the programmer to implement a ring buffer. If enabled, the ring buffer reserves a fixed range of memory to be used for a channel. The start and end address of the ring buffer are defined by the start memory address and the transfer count. As data is read from the I/O device it is written into the ring buffer in memory. When the DMA transfer has exhausted its transfer count, the channel automatically reloads the start memory address and transfer count registers and continues with the DMA transfer from the I/O device. The new data is written into memory at the start of the ring buffer, over-writing the older information that has already been read by the microprocessor. As the programmer reads information that was deposited in the ring buffer by the channel, the programmer must update the channel's stop register with the memory address of the next location that has not yet been read by the microprocessor. The stop register prevents the DMA channel from over-writing information that the microprocessor hasn't read yet.

Transfer Size

Each DMA channel can be programmed to perform either 8, 16 or 32-bit transfers.

Chapter 9

The Previous Chapter

The previous chapter, "EISA DMA," described the bus cycle types supported by the EISA DMA controller. In addition, other EISA DMA enhancements were also described.

This Chapter

In this chapter, EISA automatic system configuration is discussed. This includes a description of the slot-specific I/O address space, the EISA product identifier, and the EISA card control ports. The EISA configuration process and board description files are also covered.

The Next Chapter

The next chapter begins Part two of the book. In Part two, the Intel EISA chip set and its relationship to the major system components are discussed.

ISA I/O Address Space Problem

When the original IBM PC and XT were designed, IBM defined the use of the processor's 64KB I/O address space as shown in table 9-1.

Table 9-1. IBM PC and XT I/O Address Space Usage

I/O Address Range	Reserved For
0000h – 00FFh	256 locations set aside for I/O devices integrated onto the system board.
0100h – 03FFh	768 locations set aside for I/O expansion cards.
0400h – FFFFh	Reserved. Do not use.

I/O addresses above 03FFh could not be used due to the inadequate I/O address decode performed by many of the early I/O expansion cards. The card's I/O address decoder inspects A[9:5] to determine which of twenty-four blocks

of I/O space is currently being addressed. Each block consists of 32 locations. Figure 9-1 illustrates these twenty-four address ranges. If the currently-addressed I/O location is within the block of thirty-two locations assigned to the I/O expansion card, the card's logic examines address bits A[4:0] to determine if one of up to thirty-two I/O ports on the addressed expansion card is being addressed.

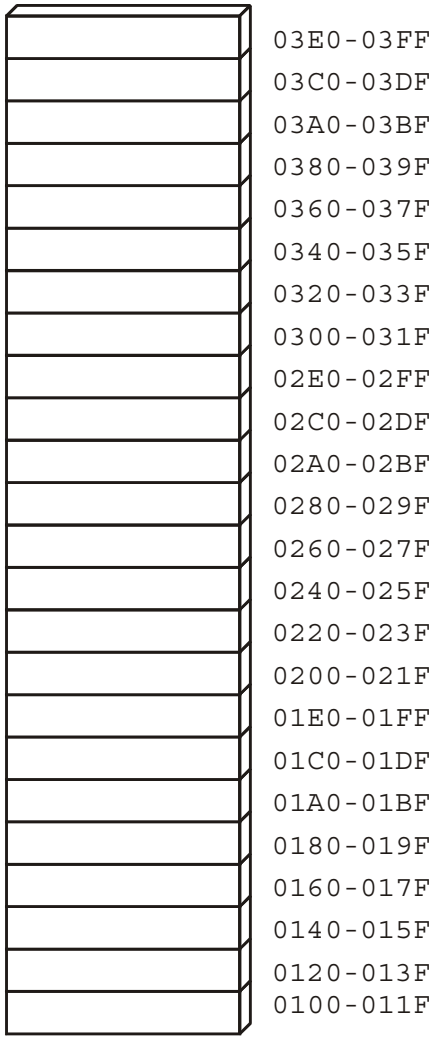


Figure 9-1. ISA Expansion I/O Ranges

The I/O address decoders on the expansion cards for the PC, XT and AT only looked at address bits A[9:5], ignoring bits A[15:10]. The I/O address range assigned for usage by expansion cards is 0100h – 03FFh. Bits A9 and A8 would therefore be either 01b (0100h – 01FFh range), 10b (0200h – 02FFh range), or 11b (0300h – 03FFh range) when an ISA I/O card is being addressed. When the microprocessor places any address within the expansion I/O address range on the address bus, an I/O expansion card may respond.

As an example, assume that a machine has two expansion cards installed. One of them performs an inadequate address decode using just A[9:5] and has eight registers residing at I/O ports 0100h – 0107h. The other card performs a full decode using A[15:5] and has four registers residing at I/O ports 0500h – 0503h. Now assume that the microprocessor initiates a one byte I/O read from I/O port 0500h. The address placed on the bus is shown in table 9-2.

Table 9-2. Example I/O Address

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

The board that occupies the 0500h – 0503h range looks at A[15:5] and determines that the address is within the 0500h – 051Fh block. It then looks at A[4:0] and determines that location 0500h is being addressed. Since this is an I/O read bus cycle, the card places the contents of location 0500h on the lower data path (this is an even address).

At the same time, the board that occupies the 0100h – 0107h range looks at A[9:5], a subset of the address seen by the other card's address decoder, and determines that the address appears to be within the 0100h – 01FFh block. It then looks at A[4:0] and determines that location 0100h is being addressed. Since this is an I/O read bus cycle, the card places the contents of location 0100h on the lower data path (this is an even address).

Since both cards are driving a byte of data onto the lower data path, SD[7:0], data bus contention is occurring. This results in garbage data and possible hardware damage because two separate current sources are driving the lower data path. The problem occurs because the card residing in the 0100h – 0107h range looks at A[9:8] and thinks that this address is within the 0100h – 01FFh range. If the card were designed to perform a full address decode using A[15:5], the problem could have been avoided.

Addresses above 03FFh may be used as long as A[9:8] are always 00b, thus ensuring that the address will not appear to be in the 0100h – 01FFh, 0200h –

02FFh, or 0300h – 03FF ranges. Table 9-3 illustrates the usability or unusability of address ranges above 03FFh.

Table 9-3. Usable and Unusable I/O Address Ranges Above 03FFh

I/O Address Range	Usable or Unusable
x000h – x0FFh	usable
x100h – x1FFh	Unusable. Appears to be 0100h – 01FFh
x200h – x2FFh	Unusable. Appears to be 0200h – 02FFh
x300h – x3FFh	Unusable. Appears to be 0300h – 03FFh
x400h – x4FFh	usable
x500h – x5FFh	Unusable. Appears to be 0100h – 01FFh
x600h – x6FFh	Unusable. Appears to be 0200h – 02FFh
x700h – x7FFh	Unusable. Appears to be 0300h – 03FFh
x800h – x8FFh	usable
x900h – x9FFh	Unusable. Appears to be 0100h – 01FFh
xA00h – xAFFh	Unusable. Appears to be 0200h – 02FFh
xB00h – xBFFh	Unusable. Appears to be 0300h – 03FFh
xC00h – xCFFh	usable
xD00h – xDFFh	Unusable. Appears to be 0100h – 01FFh
xE00h – xEFFh	Unusable. Appears to be 0200h – 02FFh
xF00h – xFFFh	Unusable. Appears to be 0300h – 03FFh

Note: where x = any hex digit

The next section describes how the EISA specification defines the usage of these allowable address ranges above 03FFh.

EISA Slot-Specific I/O Address Space

The EISA specification expands the number of I/O locations available to system and expansion board designers and also implements automatic configuration of both system and expansion boards.

In addition to the 256 I/O locations available for ISA system board I/O devices (from 0000h – 00FFh), the EISA system board has 768 additional I/O locations available for usage by system board I/O devices. Each EISA expansion slot and each embedded EISA device has 1024 locations of slot-specific I/O address space available for use (in addition to the 768 bytes of ISA I/O address space allocated to ISA expansion boards). An embedded device is an EISA I/O device that is integrated onto the motherboard. In all operational respects, it acts

as if it's installed in an EISA expansion slot. Table 9-4 defines the I/O address assignment for the EISA system board and the expansion board slots.

Table 9-4. EISA I/O Address Assignment

I/O Address Range (hex)	Reserved For	Range Reserved For
0000 – 00FF	EISA/ISA system board I/O devices	System Board
0100 – 03FF	ISA expansion cards	ISA cards
0400 – 04FF	EISA system board I/O	System Board
0500 – 07FF	alias of ISA range; do not use	
0800 – 08FF	EISA system board I/O	System Board
0900 – 0BFF	alias of ISA range; do not use	
0C00 – 0CFF	EISA system board I/O	System Board
0D00 – 0FFF	alias of ISA range; do not use	
1000 – 10FF	Slot 1 I/O	EISA slot one
1100 – 13FF	alias of ISA range; do not use	
1400 – 14FF	Slot 1 I/O	EISA slot one
1500 – 17FF	alias of ISA range; do not use	
1800 – 18FF	Slot 1 I/O	EISA slot one
1900 – 1BFF	alias of ISA range; do not use	
1C00 – 1CFF	Slot 1 I/O	EISA slot one
1D00 – 1FFF	alias of ISA range; do not use	
2000 – 20FF	Slot 2 I/O	EISA slot two
2100 – 23FF	alias of ISA range; do not use	
2400 – 24FF	Slot 2 I/O	EISA slot two
2500 – 27FF	alias of ISA range; do not use	
2800 – 28FF	Slot 2 I/O	EISA slot two
2900 – 2BFF	alias of ISA range; do not use	
2C00 – 2CFF	Slot 2 I/O	EISA slot two
2D00 – 2FFF	alias of ISA range; do not use	
repeated for every X000–XFFF range		

In order to implement the slot-specific I/O address ranges illustrated in table 9-4, the AEN logic on the system board in an ISA system must be modified. Figure 9-2 illustrates the AEN decoder located on the EISA system board.

In an ISA system, the DMAC's AEN output is connected to the AEN pin on all ISA expansion slots in parallel. During non-DMA operation, AEN is low, allowing all memory and I/O devices to decode addresses normally. When the

DMA controller, or DMAC, is bus master and is placing a memory address on the bus, it asserts AEN (Address Enable). When I/O cards detect AEN high, the DMAC is placing a memory address on the bus and the I/O cards ignore the address. When a memory card detects AEN asserted, it decodes the address on the bus to determine if the DMAC is addressing it.

In an EISA system, when the DMAC is bus master and is addressing memory, it asserts AEN, causing the system board AEN decoder (see figure 9 - 2) to assert all of its AEN outputs. Each AEN output is connected to the AEN pin on a separate connector. In this way, the AEN decoder emulates AEN operation in an ISA machine. No I/O devices should decode the address.

During non-DMA memory bus cycles, the DMAC's AEN output is deasserted, causing the system board AEN decoder to set all of its AEN outputs low. This permits all memory cards to decode addresses normally.

During non-DMA I/O bus cycles, M/IO# is low, enabling the system board AEN decoder to use the upper digit of the I/O address, A[15:12], to select which of its AEN outputs to set low. If either A8 or A9 is set to one, however, the I/O address is within the range of 768 locations set aside for ISA expansion I/O devices. The AEN decoder sets all of its AEN outputs low, allowing all of the installed I/O cards to decode the address. When a card's AEN line is sensed low, an EISA I/O device that uses slot-specific I/O address space should examine A8 and A9 to ensure both are zero before decoding A[11:0]. If either bit is high, the bus master is addressing an ISA I/O device and the EISA I/O card should not respond.

If A8 and A9 are both zero during an I/O bus cycle, the bus master is addressing slot-specific I/O address space. In response, the AEN decoder uses A[15:12] to determine which one of its AEN outputs to set low. Only the card in the expansion slot to which the selected AEN line is connected can decode and respond to the I/O address. Upon sensing its AEN line low, the card ensures that A[9:8] are zero before decoding A[11:0]. Table 9-5 defines the action taken by the system board's AEN decoder under each set of circumstances.

Table 9-5. AEN Decoder Action Table

DMAC's AEN	A9	A8	M/IO#	AEN Decoder Action
1	na	na	na	The DMAC drives its AEN output high when it is bus master and is placing a memory address on the address bus. The AEN decoder responds by driving all of its AEN outputs high. This prevents I/O devices from decoding memory addresses.
0	na	na	1	A device other than the DMAC is bus master and has initiated a memory bus cycle. In response, the AEN decoder sets all of its AEN outputs low. The low on the AEN outputs allows both memory and I/O devices to decode addresses.
0	0	0	0	A device other than the DMAC is bus master and has initiated an I/O bus cycle. Since A[9:8] are both zero, the bus master is addressing slot-specific I/O address space. In response, the AEN decoder decodes the high digit of the address, A[15:12], to determine which of its AEN outputs to set low. All of the decoder's other AEN outputs are set high. Only the I/O device in the expansion slot addressed by the high digit of the address can decode the I/O address.
0	0	1	0	The bus master is addressing an ISA I/O expansion device that resides within the 0100h – 01FFh range. In response, the AEN decoder sets all of its AEN outputs low. EISA I/O devices that use slot-specific I/O address space should not respond when either A8 or A9 are high.
0	1	0	0	The bus master is addressing an ISA I/O expansion device that resides within the 0200h – 02FFh range. In response, the AEN decoder sets all of its AEN outputs low. EISA I/O devices that use slot-specific I/O address space should not respond when either A8 or A9 are high.
0	1	1	0	The bus master is addressing an ISA I/O expansion device that resides within the 0300h – 03FFh range. In response, the AEN decoder sets all of its AEN outputs low. EISA I/O devices that use slot-specific I/O address space should not respond when either A8 or A9 are high.

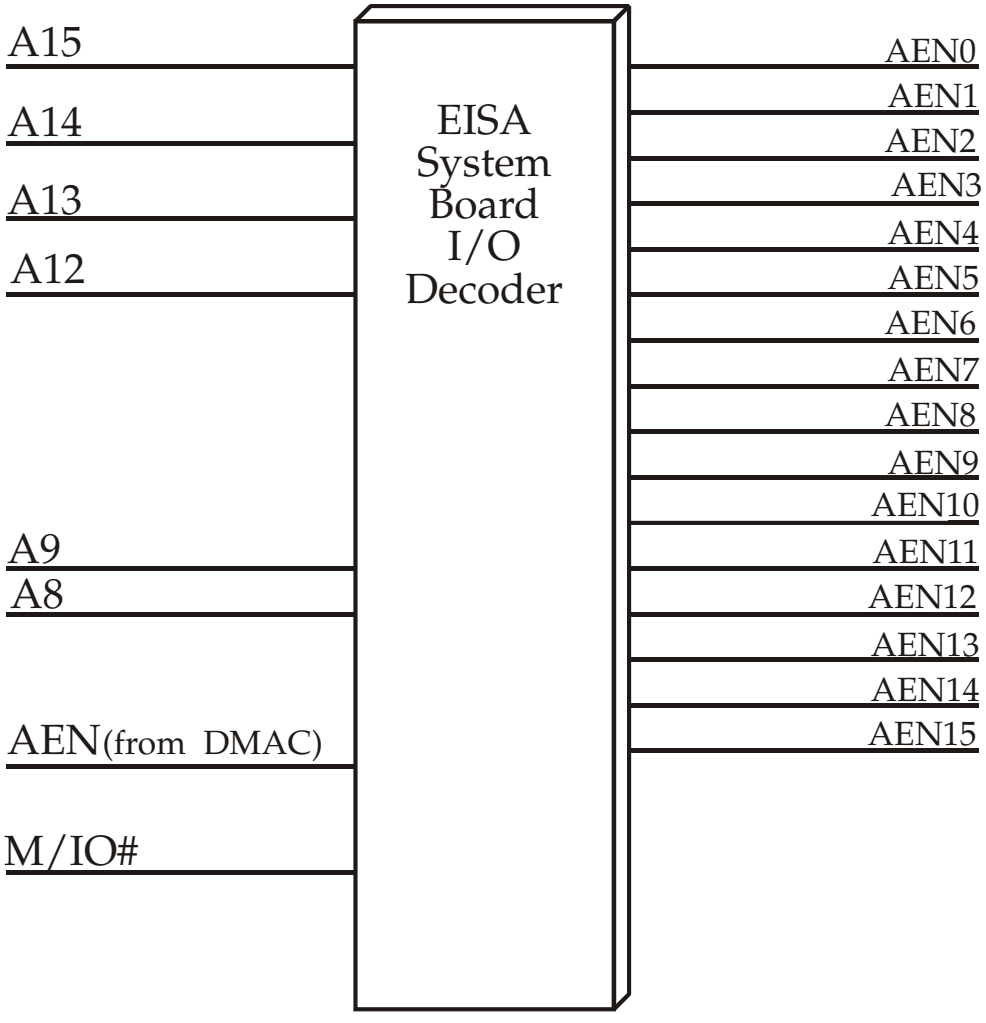


Figure 9-2. The System Board's AEN Decoder

EISA Product Identifier

EISA expansion boards, embedded devices and system boards have a four byte product ID that can be read from I/O port addresses xC80h – xC83h, where x = 0 for the system board or the number of the expansion slot the card is installed in. For example, the system board's ID can be read from I/O addresses 0C80 – 0C83h and slot one's ID can be read from 1C80 – 1C83h.

The first two bytes of the system board ID, read from I/O ports xC80 – xC81, contain a three character manufacturer's code. The three character manufacturer code is uppercase, ASCII alpha chosen by the manufacturer and registered with the firm that distributes the EISA spec. A compressed version of the ASCII code, using just the lower five bits of each character, is used. The third byte and the high-order four bits of the fourth byte are used to specify a product identifier consisting of three hex digits. The lower four bits of the fourth byte is use to specify the product revision number. Table 9-6 illustrates the format of the product ID bytes read from an expansion board. Table 9-7 illustrates the format of the product ID bytes read from an EISA system board.

To verify that an EISA expansion card is installed in a particular card slot:

- Write FFh to I/O port xC80h.
- Read one byte from xC80h.
- If the byte read equals FFh, an EISA card isn't installed in the slot. If the byte does not equal FFh and bit 7 of the byte read is zero, the card's EISA product ID can be read from xC80h – xC83h.

Table 9-6. Expansion Board Product ID Format

Location/Bits	Specify
xC80, bit 7	not used, must be 0
xC80, bits 6:2	1st compressed ASCII character of Manufacturer's ID
xC80, bits 1:0	upper two bits of 2nd compressed ASCII character of Manufacturer's ID
xC81, bits 7:5	lower three bits of 2nd compressed ASCII character of Manufacturer's ID
xC81, bits 4:0	3rd compressed ASCII character of Manufacturer's ID
xC82, bits 7:4	upper hex digit of product type
xC82, bits 3:0	middle hex digit of product type
xC83, bits 7:4	lower hex digit of product type
xC83, bits 3:0	single hex digit of product revision number

Table 9-7. EISA System Board Product ID Format

Location/Bits	Specify
0C80, bit 7	not used, must be 0
0C80, bits 6:2	1st compressed ASCII character of Manufacturer's ID
0C80, bits 1:0	upper two bits of 2nd compressed ASCII character of Manufacturer's ID
0C81, bits 7:5	lower three bits of 2nd compressed ASCII character of Manufacturer's ID
0C81, bits 4:0	3rd compressed ASCII character of Manufacturer's ID
0C82, bits 7:0	reserved for manufacturer's use
0C83, bits 7:3	reserved for manufacturer's use
0C83, bits 2:0	EISA bus version

EISA Configuration Registers

In an ISA machine, expansion cards are configured by setting DIP switches and/or jumpers to the desired settings. This allows the user to select options such as:

- the start address of a device ROM mounted on the card
- the start address of RAM located on the card
- the IRQ line the card utilizes
- the DMA channel the card utilizes
- the I/O address range the card responds to

Setting the switches and/or jumpers allows the user to resolve conflicts between installed expansion cards. In addition, many ISA system boards have switches and/or jumpers that are used to configure the system board options.

The EISA specification replaces the switches and/or jumpers with special I/O locations. Each of these I/O locations can contain up to eight bits that may be used to select options on the system or expansion card. Each I/O location may be thought of as a pseudo-DIP switch bank. They are configuration registers. These special I/O locations reside in the slot-specific I/O address space starting at xC80h and extending up to xCFFh, a total of 128 locations. The first four of these I/O locations are reserved for the card ID, while three of the eight bits in xC84h are reserved for special card functions. The remaining five bits in xC84h and locations xC85h – xCFFh are available for the implementation of card-specific configuration registers.

Configuration Bits Defined by EISA Spec

Three of the eight bits available in port xC84h must be implemented on all EISA expansion cards. Table 9-8 defines these three bits.

Table 9-8. EISA Add-in Card Configuration Bits

Port xC84	Description
bit 0	ENABLE bit. 0 = disable card; 1 = enable card. This bit is read/writable and is mandatory. Reset clears this bit to zero.
bit 1	IOCHKERR bit. This read-only bit is used to determine if am EISA card is generating CHCHK#, causing an NMI. This bit is mandatory if the card can generate CHCHK#. Reset clears this bit to zero.
bit 2	IOCHKRST bit. This write-only bit is used to reset an expansion card. Setting it high for a minimum of 500ns causes the card to be reset. When reset, the ENABLE and IOCHKERR bits are cleared and all of the card's logic is reset to an initialized state. If a card doesn't implement the IOCHKERR bit, the IOCHKRST bit need not be implemented.
bits 7:3	available for use in configuring the card.

EISA Configuration Process

General

Several elements are necessary in order to implement automatic system configuration in an EISA system. The system must have some way of verifying the placement and type of EISA boards in the system. This is accomplished by reading the board ID from each card slot during the POST.

Each EISA card must implement a set of one or more configuration registers to allow automatic configuration of the card each time the machine is powered on. The use of the configuration registers is card-specific and the registers are located in the I/O address range xC84h – xCFFh.

The manufacturer of the system board and each of the EISA and, where possible, ISA boards should supply a configuration file for each card that describes the programmable options available on the card. Programmable options might include interrupt request lines and DMA channels to be used, size and start address of required memory space and the start address of required I/O space. The configuration file must identify the options within each functional area — for example, the choice of interrupt request lines or DMA channels the card can

be configured to use. For each possible choice, the file must describe the respective bit settings and I/O port to be written in order to choose the selected option. For ISA cards, the configuration file describes the available options and the respective DIP switch and/or jumper settings necessary to implement each selected option.

The system manufacturer must provide a configuration program that is capable of examining all of the selectable options available for each of the installed cards and of producing a conflict-free scenario. In other words, it must be capable of choosing a set of options for each card where none of the selected options conflict with the option settings chosen for any other installed ISA or EISA card. The configuration program then stores the configuration information in non-volatile memory and also makes a backup copy on diskette. The diskette may then be distributed within an organization to ensure that all machines are configured the same way.

The EISA system board must incorporate at least 340 bytes of non-volatile memory for each expansion card slot and an additional 340 bytes for the system board configuration information. The block of non-volatile memory associated with a card slot is used to store card-specific configuration information such as the card ID and the address of and data to be written to the card's configuration registers each time the machine is powered on.

The system manufacturer must supply ROM-based BIOS routines that allow configuration information to be written to and read from configuration memory (non-volatile memory).

Configuration File Naming

The name of a card's configuration file consists of an exclamation point followed by the manusfacturer ID, product ID and the file extension of CFG. The following are some examples of legal configuration file names:

- !DEL1233.CFG
- !CPQ5672.CFG
- !IBM9AB1.CFG

The configuration program includes a method for handling cards with duplicate product IDs. As the configuration program copies the configuration file for each card to the configuration diskette, it checks for duplicate product IDs. When one is found, the first character of the filename is changed from an ex-

clamation point to the number one. If a third configuration file with the same product ID is found, its name is altered by changing the first character from an exclamation point to the number two, and so on. As an example, assume that the machine being configured has three boards with the same product ID. As the three configuration files are copied to the configuration diskette, they are renamed as follows:

- first file name is left as !DEL1231.CFG
- second file name is altered to 1DEL1231.CFG
- third file name is altered to 2DEL1231.CFG

The card manufacturer should always ensure that the card's configuration file name and product ID are changed to reflect the actual revision number of the card.

Configuration Procedure

The example sequence that follows provides a guide to the configuration of an EISA system.

1. With the machine powered off, insert the configuration diskette in floppy drive A.
2. Install all EISA expansion cards. Do not install ISA cards yet.
3. Power on the machine. During the POST, the machine attempts to read the product ID from each expansion slot in order to determine which slots have EISA cards installed.
4. When the POST is complete, the unit boots from the configuration diskette and executes the configuration program.
5. Use the “copy configuration file” command on the configuration program's menu to copy each of the configuration files for the installed EISA cards and the yet-to-be-installed ISA cards onto the configuration diskette. During the copy process, the configuration program automatically detects and renames the configuration files for cards with duplicate product IDs.
6. Select “automatic system configuration” from the menu. The configuration program automatically generates a conflict-free scenario for both the EISA and ISA cards. The configuration program stores the EISA card product IDs, I/O configuration port addresses and the data to be written to each configuration port in non-volatile memory. Information about the ISA cards is also stored in the slot-specific non-volatile memory areas reserved for the slots the ISA cards are to be installed in.

7. Using the prompts generated by the configuration program, the user sets the DIP switches and/or jumpers on the ISA cards to the indicated positions.
8. Print a hardcopy of the expansion slots the ISA cards must be installed in and any command lines that may need to be entered into the operating system's startup files (such as the CONFIG.SYS and AUTOEXEC.BAT files in an MS-DOS environment).
9. Turn the system off and install the ISA cards in the expansion slots indicated by the configuration program. Refer to the hardcopy.
10. Remove the configuration program diskette from drive A: and power up the system again. The system now boots from the hard disk.
11. Using a text editor, incorporate command lines into the operating system's startup files that were indicated by the configuration program. Refer to the hardcopy.
12. Reboot the system so the commands in the operating system's startup files are executed.

Configuration File Macro Language

The option information contained within a configuration file is written in a high-order macro language developed by the EISA consortium specifically for this purpose. The syntax of this language is described in detail in the EISA specification. It would be counter-productive to duplicate the entire language definition within this document. The following section provides an annotated listing of a sample configuration file.

Example Configuration File

The following example configuration file demonstrates many, but not all, of the elements found in the typical configuration file. The text following the example explains each element.

```
BOARD1
   ID = "TLC0011"
   NAME = "XYZ Corp. Ethernet Board - Rev. 5"
   MFR = "XYZ Corp."
   CATEGORY = "NET"
   SLOT = EISA
   LENGTH = 330
   READID=YES

IOPORT(1) = 0zC94h2
   INITVAL = 0000xxxx
IOPORT(2) = 0zC98h3
   INITVAL = xxxxxxxxxxxxxxrr
IOPORT(3) = 0zC9Ah4
   INITVAL = xxxxxxrr
IOPORT(4) = 0zC9Bh5
   INITVAL = rrrrrxxx
IOPORT(5) = 0zC85h6
   INITVAL = xxxxxxxx
IOPORT(6) = 0zC86h7
   INITVAL = 0rrxxxxx
IOPORT(7) = 0zC86h8
   INITVAL = 1rrxxxxx

SOFTWARE(1) = "TLCDRVR.EXE - \n If using MS-
     DOS, place the following command line in
     AUTOEXEC.BAT:\n\t\tTLCDRVR /S=n /A =n\n
     Use the following values with the /S and
     /A parameters:" 9
```

```
; Function description starts here
GROUP = "Ethernet Network Interface"10
   TYPE = "NET,ETH"11
FUNCTION = "Network Interface Location"12
   CHOICE = "Set Up as Node 0"13
      SUBTYPE = "LAN0"
      FREE
         INIT = SOFTWARE(1) = "/S = 1 /A = 0"
         INIT = IOPORT(5) = LOC (5-2) 0000
      CHOICE = "Set up as Node 1"
      SUBTYPE = "LAN1"
      FREE
         INIT = SOFTWARE(1) = "/S = 0 /A = 1"
         INIT = IOPORT(5) = LOC (5-2) 0001
      CHOICE = "Set Up as Node 2"
      SUBTYPE = "LAN2"
      FREE
         INIT = SOFTWARE(1) = "/S = 0 /A = 2"
         INIT = IOPORT(5) = LOC (5-2) 0010
      .
      .
      .
      CHOICE = "Set Up as Node 15"
      SUBTYPE = "LAN15"
      FREE
         INIT = SOFTWARE(1) = "/S = 0 /A = 15"
         INIT = IOPORT(5) = LOC (5-2) 1111
```

```
FUNCTION = "DMA and Interrupt assignment"14
   CHOICE = "System Resources"15
      ;DMA channel uses Type "C" bus cycle16
      LINK17
         DMA = 5|718
         SHARE = no
         SIZE = dword
         TIMING = TYPEC
         INIT = IOPORT(5) LOC (0) 0|1

      ;interrupt is level-sensitive, shareable
      LINK
         IRQ = 2|519
         SHARE = yes
         TRIGGER = level
         INIT = IOPORT(5) LOC (1) 0|1
      COMBINE20
      MEMORY = 2K21
      ADDRESS = 0C0000h|0D0000h|0E0000h
      MEMTYPE = oth
      WRITABLE = no
      SHARE = no
      SIZE = byte
      CACHE = yes
      DECODE = 32
      INIT=IOPORT(6)LOC(3-0) 1100|1101|1110
```

```
;network board local RAM
FUNCTION = "Local RAM initialization"22
   CHOICE = "64K RAM"23
      SUBTYPE = "64K"
      COMBINE
         MEMORY = "64K"
         ADDRESS= 100000h-1F0000h STEP = 64K
         WRITABLE = yes
         MEMTYPE = oth
         SIZE = dword
         CACHE = no
         INIT=IOPORT(7)LOC(4 3 2 1 0)00000-01111
   CHOICE = "128K RAM"24
      SUBTYPE = "128K"
      COMBINE
         MEMORY = "128K"
         ADDRESS = 100000h-1F0000h STEP = 64K
         WRITABLE = yes
         MEMTYPE = oth
         SIZE = dword
         CACHE = no
         INIT=IOPORT(7)LOC(4 3 2 1 0)10000-11111
ENDGROUP25
```

```
;serial port section
FUNCTION = "Serial Port"26
   TYPE = "COM,ASY"27
   CHOICE = "COM1"28
      SUBTYPE = "COM1"
      FREE
      IRQ = 4
         SHARE = yes
         TRIGGER = level
      PORT = 3F8h-3FFh
         SHARE = no
         SIZE = byte
      INIT = IOPORT(1) LOC (3-0) 0000
      INIT=IOPORT(2)LOC (15-2) 00000011111100
      INIT = IOPORT(3) LOC (7-2) 110000
      INIT = IOPORT(4) LOC (2-0) 010
   CHOICE = "COM2"29
      SUBTYPE = "COM2"
      FREE
      IRQ = 3
         SHARE = yes
         TRIGGER = level
      PORT = 2F8h-2FFh
         SHARE = no
         SIZE = byte
      INIT = IOPORT(1) LOC (3-0) 0000
      INIT = IOPORT(2)LOC (15-2) 00000011111100
      INIT = IOPORT(3) LOC (7-2) 110000
      INIT = IOPORT(4) LOC (2-0) 000

   CHOICE = "Serial Port Disable"30
      SUBTYPE = "Port Disable"
      DISABLE = yes
      FREE
         INIT = IOPORT(4) LOC (0) 0
```

Example File Explanation

Each of the numbered sections that follow provides an explanation of the section of the example configuration file with the corresponding subscripted number.

1. Every configuration file must include the board identification block. The BOARD statement identifies the beginning of the block. The ID statement contains the product ID consisting of the three character manufacturer's code, the three digit board type and the one digit revision number. The NAME field contains text that describes the board. The MFR field contains the full name of the board manufacturer. The CATEGORY field contains a three character designator that identifies the basic board type. Table 9-9 provides a listing of the available categories. The SLOT statement identifies the type of slot the board requires. If the SLOT statement is missing, the configuration program assumes that the board requires a 16-bit ISA slot. The LENGTH statement specifies the length of the board in millimeters. The READID statement identifies whether the board has a product ID that can be read from I/O ports xC80h – xC83h.
2. The IOPORT(1) statement associates the variable name IOPORT(1) with I/O port address xC94h. The INITVAL statement identifies the source of each of the bits within the specified I/O port. In this example statement, the xxxx indicates that bits 3:0 are supplied by the configuration program based on the configuration chosen. The 0000 in bits 7:4 indicates that these bits are always zero.
3. The IOPORT(2) statement associates the variable name IOPORT(2) with I/O port addresses xC98h and xC99h. The INITVAL statement identifies the source of each of the bits within the specified I/O port. In this example statement, the bit field is sixteen bits wide, indicating that this is a 16-bit I/O port. Bits 1:0 have an “rr” designation, meaning that they are read-only bits. The x's in bits 15:2 indicate that they are supplied by the configuration program based on the configuration chosen.
4. The IOPORT(3) statement associates the variable name IOPORT(3) with I/O port address xC9Ah. The INITVAL statement identifies the source of each of the bits within the specified I/O port. In this example statement, the bit field is eight bits wide, indicating that this is an 8-bit I/O port. Bits 1:0 have an “rr” designation, meaning that they are read-only bits. The x's in bits 7:2 indicate that they are supplied by the configuration program based on the configuration chosen.
5. The IOPORT(4) statement associates the variable name IOPORT(4) with I/O port address xC9Bh. The INITVAL statement identifies the source of each of the bits within the specified I/O port. In this example statement,

the bit field is eight bits wide, indicating that this is an 8-bit I/O port. Bits 7:3 have an "r" designation, meaning that they are read-only bits. The x's in bits 2:0 indicate that they are supplied by the configuration program based on the configuration chosen.

6. The IOPORT(5) statement associates the variable name IOPORT(5) with I/O port address xC85h. The INITVAL statement identifies the source of each of the bits within the specified I/O port. In this example statement, the bit field is eight bits wide, indicating that this is an 8-bit I/O port. The x's in bits 7:0 indicate that they are supplied by the configuration program based on the configuration chosen.
7. The IOPORT(6) statement associates the variable name IOPORT(6) with I/O port address xC86h. The INITVAL statement identifies the source of each of the bits within the specified I/O port. In this example statement, the bit field is eight bits wide, indicating that this is an 8-bit I/O port. Bit seven is always zero. Bits 6:5 have an "r" designation, meaning that they are read-only bits. The x's in bits 4:0 indicate that they are supplied by the configuration program based on the configuration chosen.
8. The IOPORT(7) statement associates the variable name IOPORT(7) with I/O port address xC86h. The INITVAL statement identifies the source of each of the bits within the specified I/O port. In this example statement, the bit field is eight bits wide, indicating that this is an 8-bit I/O port. Bit seven is always one. Bits 6:5 have an "r" designation, meaning that they are read-only bits. The x's in bits 4:0 indicate that they are supplied by the configuration program based on the configuration chosen.
9. The SOFTWARE(1) statement provides the end user with instructions regarding a customized command line to be written into operating system startup files like AUTOEXEC.BAT and/or CONFIG.SYS. Customization of the command line is based on selections made during the configuration process. The text located within the quotes will be displayed for the end user and may be printed out as well. The "\n" will cause the configuration program to output a "new line" to the screen, while the "\t" represents a tab.
10. The GROUP statement block begins with the GROUP statement and ends with the ENDGROUP statement. The option choices for board functions that may be logically grouped are placed within the GROUP block. In this example, the network card being described contains both a network interface and a serial port. All of the card's functions related to the network interface are grouped together.
11. The TYPE and SUBTYPE identifiers are used by device drivers to identify, set up and operate a device that is compatible with the device driver. In the example, NET indicates it is a network interface and ETH indicates that it is an Ethernet network interface.

12. The FUNCTION statement provides the name of the functional area to be configured. In this example, it is the location of the network interface on the network.
13. The statements within a CHOICE block define the option settings for a given choice. The first CHOICE block specifies the most desired choice, with subsequent choices in order to preference. In the example, the first CHOICE block defines the settings if the network interface board is to be configured as node 0 on the network. If this choice is made, the SUBTYPE field is set to "LAN0" to supply additional information to the card's software driver. The elements with a free-form group, defined by the FREE statement, have no functional relationship to each other. The first INIT statement declares that the text string "/S = 1 /A = 0" will be appended to the text in the SOFTWARE(1) variable if the first choice is selected. In addition, the second INIT statement declares that bits 5:2 of the I/O port specified in the variable IOPORT(5), port xC85h, must be set to zero to configure the network interface card as node 0 on the network.
14. The next functional area to be configured is the assignment of the interrupt request line, DMA channel and the start address of the card's device ROM.
15. There is only one CHOICE block within this functional area.
16. This is a comment line.
17. The elements of a LINK group have a direct relationship to each other. The first LINK block contains statement relating to the DMA channel selection and programming. The second LINK block contains statements relating to the selection and programming of an interrupt request line.
18. The DMA statement offers a choice of DMA channel five or seven. The vertical line between the two numbers is the logical "or" symbol. The SHARE statement declares the DMA channel as not shareable. The SIZE statement declares the DMA channel as handling doubleword, or 32-bit, transfers. The TIMING statement declares that the selected DMA channel must be programmed to use Type "C" bus cycles. The INIT statement declares that bit zero of IOPORT(5), port xC85h, must be set to zero to select DMA channel five or to one to select DMA channel seven.
19. The second LINK block contains statements relating to the selection and programming of an interrupt request line. It allows a choice of IRQ two or five, the selected IRQ input must be programmed as a shareable, level-sensitive interrupt request line, and IOPORT(5), port xC85h, bit one must be set to zero to select IRQ2, or to one to select IRQ5.
20. The elements of a combined group have an indirect relationship to each other.
21. The MEMORY statement identifies the start of a memory description block. This block describes a block of memory 2K in size. The ADDRESS statement provides a choice of one of three possible start addresses for the

memory block. The three possible start addresses are 0C0000h, 0D0000h, or 0E0000h. The MEMTYPE field identifies whether the memory block is normal system memory (SYS), expanded memory (EXP), a LIM page frame (VIR), or memory space used for memory-mapped I/O or bank-switched memory (OTH for other). OTH is primarily intended for memory-mapped I/O devices such as network cards. This memory block is declared not writable (WRITABLE = no), meaning it is ROM memory. The memory block may not be shared with another device (SHARE = no). It is 8-bit memory (SIZE = byte). It is safe to cache information from this area of memory (CACHE = yes). All 32 address lines are decoded by the board (DECODE = 32). To implement the selected memory start address, IOPORT(6), port xC86h, bits 3:0, must be set to Ch (1100), Dh (1101), or Eh(1110).

22. The next functional area to be configured is the RAM memory residing on the network interface card.
23. The first CHOICE block defines the configuration if the network interface card has 64K of RAM memory installed. Its SUBTYPE is declared as 64K for the use of the network interface driver. If this choice is made, the MEMORY block statement declares the memory as 64K in size. Its start address may begin on any one of sixteen possible address boundaries within the 1M range between 100000h and 1FFFFFh and the must start at an address divisible by 64K. It is declared as writable, meaning it is RAM memory that can be both written to and read from. It is declared with a MEMTYPE of OTH. It is a 32-bit device and the selected memory address range is declared as non-cacheable. If this choice is made, IOPORT(7), port xC86h, bits 4:0 must be set to a value between 0 0000 and 0 1111, depending on the start address selected.
24. The statements within the second CHOICE block will be executed if the network interface card has 128K of RAM memory installed. The setup is the same as that with 64K of RAM installed except for the SUBTYPE declaration and the value to be written to IOPORT(7), port xC86h. If this choice is made, IOPORT(7), port xC86h, bits 4:0 must be set to a value between 1 0000 and 1 1111, depending on the start address selected.
25. The ENDGROUP statement marks the end of the network interface portion of the configuration information. The remaining configuration information relates to the serial port.
26. The next functional area to be configured is the serial port logic residing on the network interface card.
27. For the benefit of the device driver software, the SUBTYPE is declared as "COM,ASY" meaning asynchronous communications port.
28. There are three possible configuration choices for the serial port: COM1, COM2, or disabled. For the COM1 choice, the following selections are made: the serial port will use IRQ4 and it will be programmed as a share-

able, level-triggered IRQ input; it will respond to port addresses 03F8h – 03FFh; and its I/O ports may not be shared by another device and they are 8-bit ports. Bits 3:0 of IOPORT(1), port xC94h, will be set to zeros. Bits 1:0 and 15:8 of IOPORT(2), ports xC98h and xC99h, will be set to zeros, while bits 7:2 will be set to ones. Bits 5:2 of IOPORT(3), port xC9Ah, will be set to zeros, while bits 7:6 will be set to ones. Bits 0 and 2 of IOPORT(4), port xC9Bh, will be set to zero, while bit 1 is set to one.

29. For the COM2 choice, the selections made are the same as COM1, except: the serial port will use IRQ3; and it will respond to port addresses 02F8h – 02FFh. Bits 2:0 of IOPORT(4), port xC9Bh, will be set to zero.
30. If the serial port is to be disabled, bit 0 of IOPORT(4), port xC9Bh, is set to zero and the SUBTYPE is set to disabled for the driver.

Table 9-9. Category List

Category Name	Description
COM	communications device
KEY	keyboard
MEM	memory board
MFC	multifunction board
MSD	mass storage device
NET	network board
NPX	numeric coprocessor
OSE	operating system environment
OTH	other
PAR	parallel port
PTR	pointing device
SYS	system board
VID	video board

PART TWO

THE INTEL 82350DT CHIP SET

Chapter 10

The Previous Chapter

In the previous chapter, automatic system configuration was described.

This Chapter

This chapter describes the major buses found in virtually all EISA systems. This includes the host, EISA, ISA and X-buses.

The Next Chapter

The next chapter, “Bridge, Translator, Pathfinder, Toolbox,” describes the major functions provided by the EISA chipset.

Introduction

Refer to figure 10-1. EISA systems may incorporate a number of buses such as:

- Host bus
- EISA bus
- X-bus
- Local bus

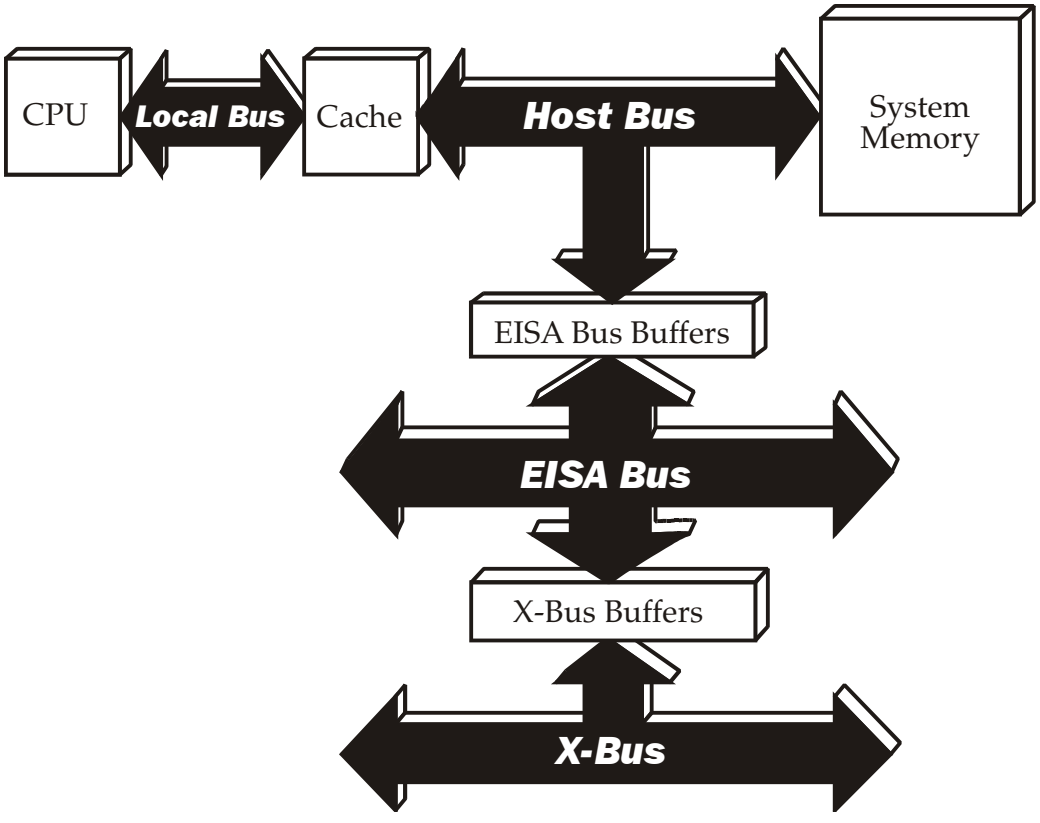


Figure 10-1. Buses Typically Found in EISA Systems

Host Bus

Virtually all EISA systems are shipped with an integral CPU. This CPU may be integrated onto the system board itself or may reside on a special, CPU daughter card that installs in a special connector on the system board. This is referred to as the host CPU. The host CPU's local address, data and control buses comprise the host bus. Typically, devices that the CPU requires fast access to would be placed on the host bus. These would include devices like:

- system board RAM memory
- numeric coprocessor
- local cache controller and cache memory
- non-cacheable access, or NCA, logic

- advanced video controller
- other I/O devices requiring fast access to the CPU

If the host CPU resides on a daughter card, the CPU's local cache controller, cache memory, NCA logic and numeric coprocessor also typically reside on the CPU card.

EISA/ISA Bus

Since the ISA bus is a subset of the EISA bus, any reference to the EISA bus in this book is a reference to the ISA bus and its EISA extensions. The ISA bus is discussed in detail in the MindShare book entitled *ISA System Architecture*. The EISA extensions to the ISA bus are described earlier in this book.

X-Bus

The ability of the microprocessor to drive data onto the data bus and the address onto the address bus is limited by the power of its output drivers. When the microprocessor is writing data to any external memory or I/O device, the data is driven out onto the processor's local data bus. If the local data bus is fanned out and connected to too many external devices, the drive capability of the microprocessor's output drivers may be exceeded and the data driven onto the data bus becomes corrupted. The local data bus is connected to the external data bus transceivers pictured in figure 10-2.

During a write operation, the bus control logic allows the appropriate data bus transceiver to pass data from the processor's local data bus onto the system data (SD) bus. The output drive capability of the transceiver is substantially greater than that of the processor's internal drivers, allowing the SD bus to fan out to more places. The SD bus is connected to all of the ISA expansion slots. In addition, many devices that may be written to are physically located on the system board itself. However, it would exceed the output drive capability of the data bus transceivers to fan out the SD bus to all of the devices integrated onto the system board as well as to all of the expansion slots.

To solve this problem, the SD bus is passed through another transceiver onto the XD, or extended data, bus. The X data bus transceiver redrives the data onto the XD bus during writes, permitting the data to be fanned out the devices residing on the XD bus. The devices integrated onto the system board are connected to the X data bus.

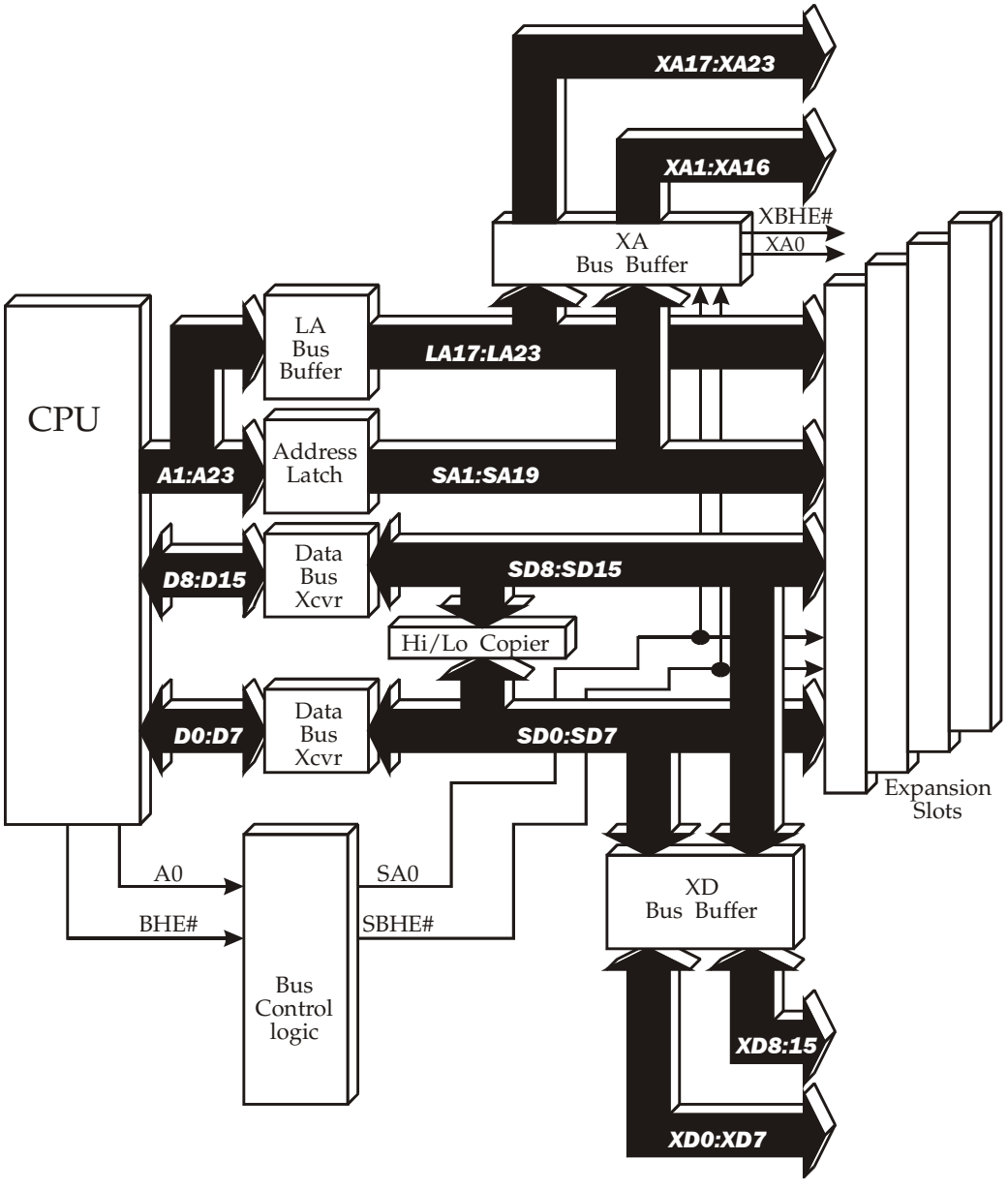


Figure 10-2. The X-Bus

When a write is in progress, the bus control logic sets up the data bus transceivers to pass data from the microprocessor's local data bus onto the SD bus and also enables the X data bus transceiver to pass data from the SD bus to the XD bus. When a read is in progress, the bus control logic sets up the X data bus transceiver to pass data from the XD to the SD bus and sets up the data bus transceivers to pass data from the SD to the microprocessor's local data bus. It should be noted that the XD bus is just a buffered version of the ISA bus's SD bus.

The same fanout problem exists on the processor's address bus. The address generated by the microprocessor is driven onto the processor's local address bus. In an ISA machine, it then passes through the LA bus buffer, the address latch and the bus control logic onto the ISA address bus. The ISA address bus consists of LA[23:17], SA[19:0] and SBHE#. The redrive capability of the LA bus buffer, the address latch and the bus control logic permits the address information to be fanned out to all of the ISA expansion slots. In addition to the ISA devices installed in expansion slots, however, the address information must also be fanned out to the addressable devices that are integrated onto the system board. This would exceed the drive capability of the LA bus buffer, the address latch and the bus control logic. To allow additional fanout, the ISA address information is passed through a buffer onto the XA bus. The buffer's redrive capability permits the XA address to be fanned out to all the devices integrated onto the system board. In other words, the devices integrated onto the system board are connected to the XA and XD buses, a buffered version of the ISA address bus.

Chapter 11

The Previous Chapter

The previous chapter introduced the buses around which all EISA systems are constructed. They are the host, EISA, ISA and X buses.

This Chapter

This chapter provides a description of the major functions performed by the EISA chipset. It acts as the bridge between the host and EISA buses. It translates addresses and other bus cycle information into a form understood by all of the host, EISA, ISA and X-bus devices in a system. When necessary, it performs data bus steering to ensure data travels over the correct paths between the current bus master and the currently-addressed device. It incorporates a toolbox including all of the standard support logic necessary in any EISA machine. It should be noted that the ISA bus is a subset of the EISA bus. For this reason, all references to the EISA bus in this or any other MindShare book refer to both the ISA bus and the Extended ISA bus (EISA).

The Next Chapter

The next chapter, "Intel 82350DT EISA Chipset," provides an introduction to Intel's EISA chipset.

Bus Cycle Initiation

When a device requires the use of the bus to communicate with another device in the system, it requests the use of the bus from the CAC. Upon being granted ownership of the bus, the bus master initiates the bus cycle by addressing the target device, or slave.

Bridge

Upon sensing the start of the bus cycle, the EISA chipset must aid in the communication process. Acting as a bridge, the EISA chipset must allow the address generated by the bus master to propagate onto all of the system buses so all of the devices in the system have an opportunity to determine if they are currently being addressed. In this section, this function is referred to as bridging. This term isn't part of the EISA specification, but is employed here to reinforce the visual image of the process being described. Table 11-1 defines the circumstances under which the EISA chipset must act as a bridge. Figure 11-1 illustrates the relationship of the bridge to the three buses. At the start of a bus cycle, neither the current bus master nor the EISA chipset knows which bus the target slave is located on. For this reason, the EISA chipset always propagates addresses generated by the host CPU onto the EISA and X-buses. Conversely, it always propagates addresses by an EISA or ISA bus master onto the host and X-buses.

Table 11-1. Situations Requiring Address Bridging

Bus Master Type	Slave Type	Action Required
Host CPU	host slave	No bridging required.
Host CPU	EISA slave	Address must be passed from the host bus to the EISA bus.
Host CPU	ISA expansion slave	Address must be passed from the host bus onto the ISA bus.
Host CPU	ISA X-bus slave	Address must be passed from the host bus onto the ISA bus and then onto the X-bus.
EISA Bus Master	host slave	Address must be passed from the EISA bus to the host bus.
EISA Bus Master	EISA slave	No bridging required.
EISA Bus Master	ISA expansion slave	No bridging required.
EISA Bus Master	ISA X-bus slave	Address must be passed from the EISA bus to the X-bus.
ISA Bus Master	host slave	Address must be passed from the ISA bus to the host bus.
ISA Bus Master	EISA slave	No bridging required.
ISA Bus Master	ISA expansion slave	No bridging required.
ISA Bus Master	ISA X-bus slave	Address must be passed from the ISA bus to the X-bus.

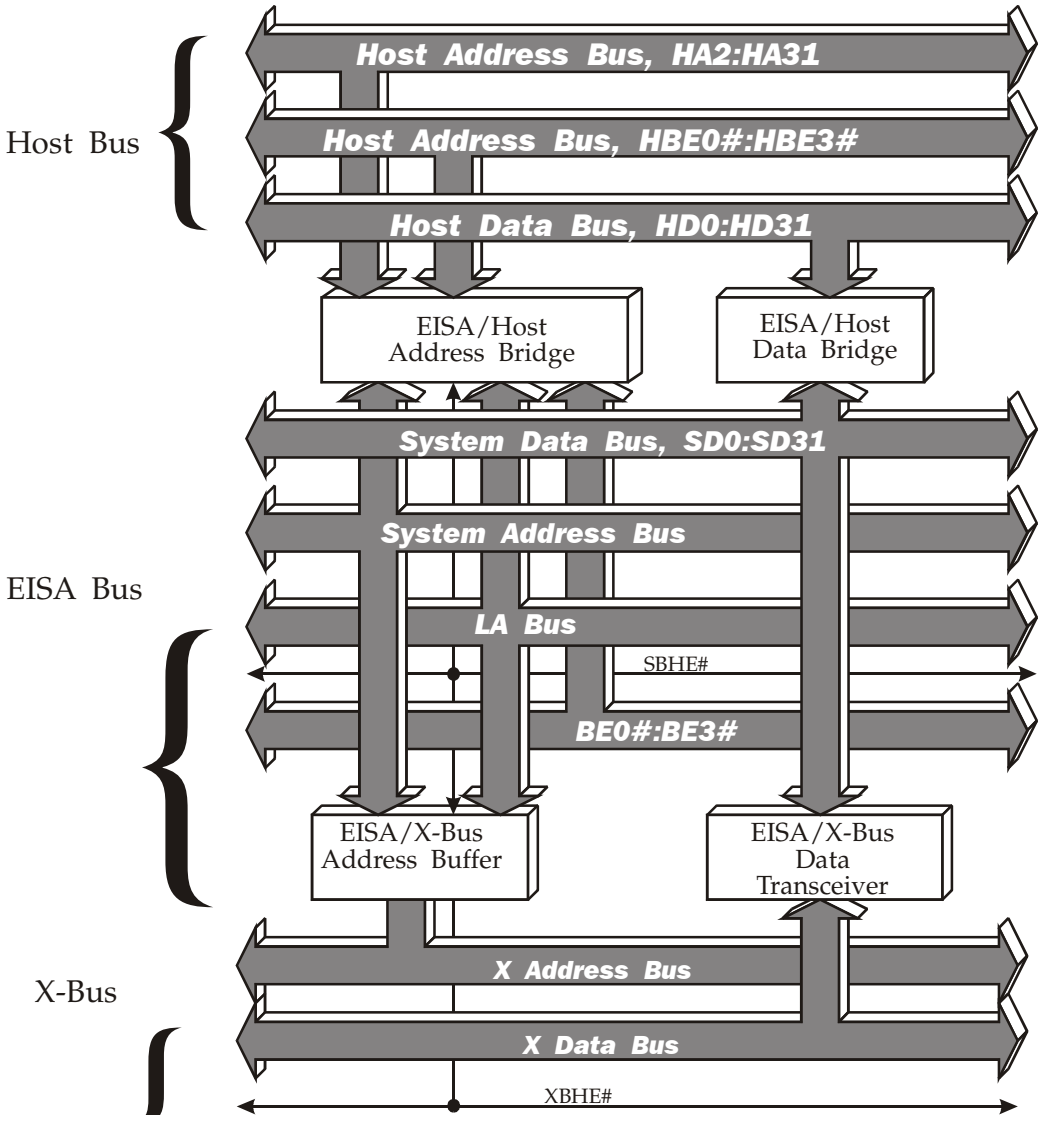


Figure 11-1. The Bridge

In addition, under some circumstances the data being transferred between the bus master and the slave must be allowed to pass from one system bus to another. Table 11-2 defines these situations.

Table 11-2. Situations Requiring Data Bridging

Bus Master Type	Slave Type	Action Required
Host CPU	host slave	No bridging required.
Host CPU	EISA slave	On a read, data must be passed from the EISA data bus to the host data bus. On a write, data must be passed from the host data bus to the EISA data bus.
Host CPU	ISA expansion slave	On a read, data must be passed from the ISA data bus to the host data bus. On a write, data must be passed from the host data bus to the ISA data bus.
Host CPU	X-bus slave	On a read, data must be passed from the X data bus to the ISA data bus and then from the ISA data bus to the host data bus. On a write, data must be passed from the host data bus to the ISA data bus and then to the X data bus.
EISA Bus Master	host slave	On a read, data must be passed from the host data bus onto the EISA data bus. On a write, data must be passed from the EISA data bus to the host data bus.
EISA Bus Master	EISA slave	No bridging required.
EISA Bus Master	ISA expansion slave	No bridging required.
EISA Bus Master	X-bus slave	On a read, data must be passed from the X data bus to the EISA data bus. On a write, data must be passed from the EISA data bus to the X data bus.
ISA Bus Master	host slave	On a read, data must be passed from the host data bus to the ISA data bus. On a write, data must be passed from the ISA data bus to the host data bus.
ISA Bus Master	EISA slave	No bridging required.
ISA Bus Master	ISA expansion slave	No bridging required.
ISA Bus Master	X-bus slave	On a read, data must be passed from the X data bus to the ISA data bus. On a write, data must be passed from the ISA data bus to the X data bus.

Translator

Address Translation

The EISA chipset must translate the address being generated by the bus master to forms that are understood by the slave devices on all three buses. Table 11-3 defines the different forms of address information expected by devices on the three buses.

Table 11-3. Address Translation Table

Bus Master		Slave Type and Address Expected				
Type	**Address**	**8-bit ISA**	**16-bit ISA**	**16-bit EISA**	**32-bit EISA**	**32-bit Host**
Host CPU	A[31:2] and BE#[3:0]	SA[19:0]	SA[23:0] and SBHE#	LA[31:2] and BE#[3:0]	LA[31:2] and BE#[3:0]	A[31:2] and BE#[3:0]
16-bit EISA Bus Master	LA[31:2] and BE#[3:0]	SA[19:0]	SA[23:0] and SBHE#	LA[31:2] and BE#[3:0]	LA[31:2] and BE#[3:0]	A[31:2] and BE#[3:0]
32-bit EISA Bus Master	LA[31:2] and BE#[3:0]	SA[19:0]	SA[23:0] and SBHE#	LA[31:2] and BE#[3:0]	LA[31:2] and BE#[3:0]	A[31:2] and BE#[3:0]
16-bit ISA Bus Master	SA[23:0] and SBHE#	SA[19:0]	SA[23:0] and SBHE#	LA[31:2] and BE#[3:0]	LA[31:2] and BE#[3:0]	A[31:2] and BE#[3:0]

When an EISA bus master or the host CPU is performing a bus cycle, the EISA chipset must convert the bus master's byte enable outputs, BE#[3:0], into the correct setting on the A0, A1 and BHE# signal lines. Conversely, when an ISA bus master is performing a bus cycle, A0, A1 and BHE# must be converted to the correct setting on the byte enable lines.

Command Line Translation

Each of the three types of bus masters, EISA, ISA and host CPU, uses a specific set of signal lines to indicate the address phase and data phase periods and the type of bus cycle in progress. Conversely, each of the three types of slaves recognizes the same respective set of signals indicating address phase, data phase and the bus cycle type. When a bus master initiates a bus cycle, the EISA chipset must convert the bus master's signal set to those recognized by the other

two slave types. This enables any bus master type to communicate with devices of any other type. Table 11-4 indicates the signal lines used by each of the three bus master and slave types to indicate address phase, data phase and the bus cycle type.

Table 11-4. Command Lines

Device Type	Address Phase Signal	Data Phase Signal	Bus Cycle Type Indicators
EISA	START#	CMD#	M/IO# and W/R#
ISA	BALE	SMRDC#, SMWTC#, MRDC#, MWTC#, IORC#, IOWC#	SMRDC#, SMWTC#, MRDC#, MWTC#, IORC#, IOWC#
Host	ADS#	End of ADS# until READY# sampled active	W/R#, M/IO# and D/C#

Pathfinder

Under some circumstances, data path steering is necessary. When a bus master is communicating with a slave using a data path or paths that the slave is incapable of using, the data bus steering logic must be activated. During a read bus cycle, the data bus steering logic ensures that the returning data arrives at the bus master on the expected data path(s). During a write bus cycle, the data bus steering logic ensures that the data being written by the bus master is routed to the data path(s) that the slave expects to receive the data on. In an EISA machine, the data bus steering function is provided by the EISA chipset. Table 11-5 defines the situations when data bus steering is necessary. A more detailed description of data bus steering may be found in the MindShare book entitled *ISA System Architecture*. The 32-bit bus master or a 16-bit EISA bus master indicates the data path(s) to be used during a bus cycle using its byte enable outputs, BE#[3:0]. A 16-bit ISA bus master uses A0 and BHE# to indicate the data path(s) that will be used during a bus cycle. The addressed slave indicates the data path(s) that it is connected to by asserting IO16#, M16#, EX16# or EX32#. If IO16#, M16# or EX16# is asserted by the currently-addressed slave, it is a 16-bit device and is connected to data paths 0 and 1. If the currently-addressed slave asserts EX32#, it is a 32-bit device and is connected to all four data paths. If none of these lines are asserted, the addressed slave is an 8-bit device and is connected only to path 0.

Table 11-5. Situations Requiring Data Bus Steering

Bus Master Type	Slave Type	Bus Cycle Type	Steering Action Required
32-bit	8-bit	write	When a 32-bit bus master is writing a single byte to an 8-bit device over paths 1, 2, or 3, the data bus steering logic must copy the byte down to path 0 so it can get to the 8-bit device. When a 32-bit bus master is writing multiple bytes to an 8-bit device in a single bus cycle, the data bus steering logic must route the data to path 0 one byte at a time. As each byte is routed to the lower data path, the address seen by the 8-bit device must be incremented by the steering logic and the MWTC# or IOWC# line must be turned off and then on again to trick the 8-bit device into thinking another bus cycle has been initiated.
32-bit	8-bit	read	When a 32-bit bus master is reading a single byte from an 8-bit device over path 1, 2, or 3, the data bus steering logic must copy the byte from path 0 to the path the bus master expects to receive the byte on. When a 32-bit bus master is attempting to read multiple bytes from an 8-bit device in a single bus cycle, the 8-bit device can only return one byte at a time. The steering logic must address each byte individually, copy it to the proper data path and latch it in a latching data bus transceivers until all of the requested bytes have been retrieved. As each byte is routed to and latched by the proper data bus transceiver, the address seen by the 8-bit device must be incremented by the steering logic and the MRDC# or IORC# line must be turned off and then on again to trick the 8-bit device into thinking another bus cycle has been initiated.
32-bit	16-bit	write	When a 32-bit bus master is writing one or two bytes to a 16-bit device over paths 2 or 3, the data bus steering logic must copy the byte or bytes to path 0 and/or path 1 so that they can get to the 16-bit device.

Table 11 - 5 cont.

Bus Master Type	Slave Type	Bus Cycle Type	Steering Action Required
32-bit	16-bit	read	When a 32-bit bus master is reading one or two bytes from a 16-bit device over paths 2 or 3, the data bus steering logic must copy the byte or bytes from path 0 and/or path 1 to path 2 and/or path 3 so that they are received by the bus master over the expected data path(s).
32-bit	32-bit	read or write	none
16-bit	8-bit	write	When a 16-bit bus master is writing a single byte to an 8-bit device over path 1, the data bus steering logic must copy the byte down to path 0 so that it can get to the 8-bit device. When a 16-bit bus master is writing two bytes to an 8-bit device in a single bus cycle, the data bus steering logic must route the data to path 0 one byte at a time. As each byte is routed to the lower data path, the address seen by the 8-bit device must be incremented by the steering logic and the MWTC# or IOWC# line must be turned off and then on again to trick the 8-bit device into thinking another bus cycle has been initiated.
16-bit	16-bit	read or write	none
16-bit	32-bit	write	When a 16-bit bus master is writing one or two bytes to either of the last two locations in a doubleword in a single bus cycle, the steering logic must copy the byte or bytes to path 2 and/or path 3 so that the data will be routed to the proper location(s) within the addressed doubleword.
16-bit	32-bit	read	When a 16-bit bus master is reading one or two bytes from either of the last two locations in a doubleword in a single bus cycle, the steering logic must route the byte or bytes from path 2 and/or path 3 to path 0 and/or path 1 so that the data will be received over the proper path(s).

Toolbox

In addition to providing the bridge, translation and data bus steering functions, the EISA chipset includes a toolbox with all of the basic support elements necessary for the proper function of any EISA system. These include:

- Two modified Intel 8259A programmable interrupt controllers in a master/slave configuration
- Two modified Intel 8237 DMA controllers in a master/slave configuration
- The refresh logic
- The central arbitration control
- Five programmable timers
- The NMI logic

Detailed descriptions of interrupts, DMA, refresh, the timers and the NMI logic can be found in the MindShare book entitled *ISA System Architecture.* Information regarding the EISA-specific enhancements to the interrupt, DMA, refresh and the NMI control logic can be found earlier in this book. Information regarding the Central Arbitration Control (CAC) can be found earlier in this publication. A description of the Intel 82357 Integrated Systems Peripheral (ISP) can be found in the next chapter. The ISP, part of the Intel EISA chipset, contains all of the above-mentioned logic elements.

Chapter 12

The Previous Chapter

The previous chapter described the major functions performed by an EISA chipset.

This Chapter

This chapter provides an introduction to the Intel 82350DT EISA chipset. The focus is on the 82358DT EISA Bus Controller (EBC), the 82357 Integrated Systems Peripheral (ISP), and the 82352 EISA Bus Buffers (EBBs).

Introduction

This chapter is not intended as a substitute for the Intel publication that describes the 82350DT EISA chipset. It is intended as a companion to the Intel document, providing an introduction to the roles each component plays in a typical EISA system. Only the crucial chipset components are represented here: the EBC, the address EBB, the data EBB and the ISP. For detailed information, refer to the Intel document entitled "82350DT EISA Chipset," order number 290377-002. The EBC can be configured to operate in three different types of environments:

- With the host interface unit interfaced directly to the host CPU subsystem. This is referred to as the 82350 environment.
- With the host interface unit interfaced to the host bus through the Intel 82359 DRAM controller. This is referred to as the 82350DT/enhanced environment.
- With the host interface unit interfaced to a buffered bus. The buffered bus, in turn, is connected to the Intel 82359 DRAM controller, which is connected to the host bus. This is referred to as the 82350DT/buffered environment.

This chapter describes operation of the EISA chipset configured for the 82350 environment.

Figure 12-1 illustrates the relationship of the Intel EBC, ISP, Data Buffer and Address Buffer to the host, EISA/ISA and X-buses in the 82350 environment.

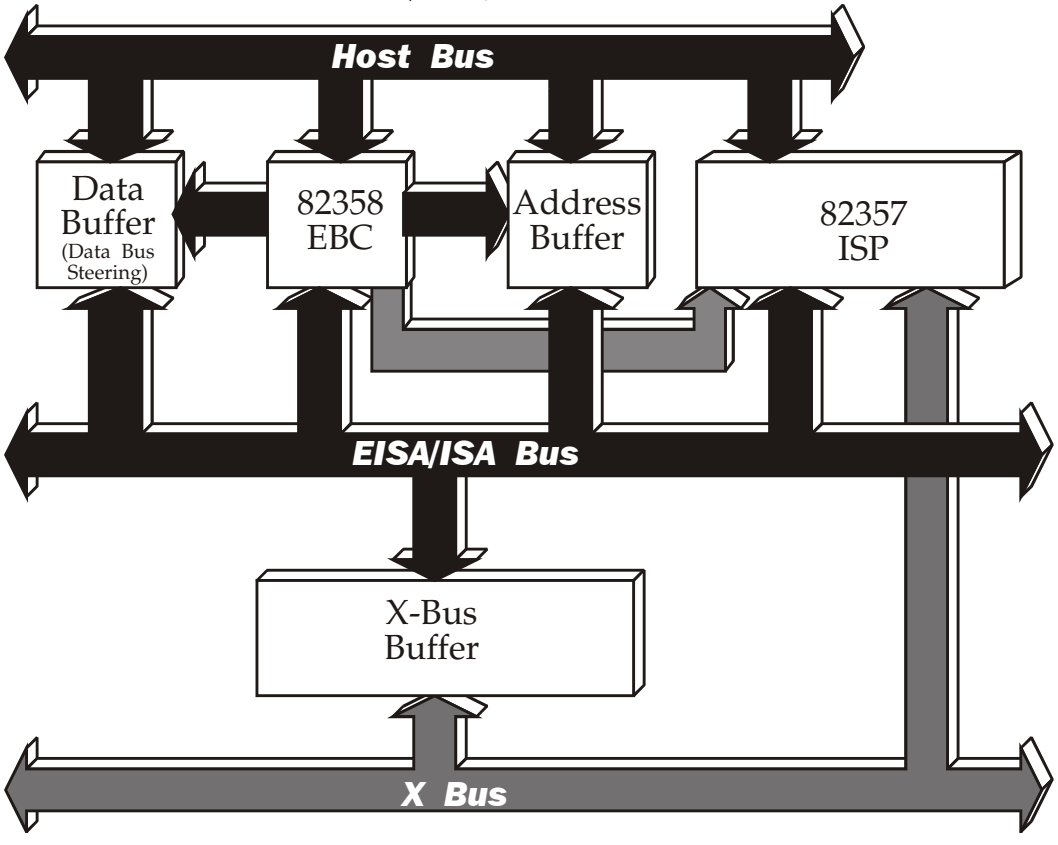


Figure 12-1. The Intel EISA Chipset

EISA Bus Controller (EBC) and EISA Bus Buffers (EBBs)

General

The EBC is pictured in figure 12-2. Together with the Data and Address EBBs, the EBC provides the bridging, translation and data bus steering functions described in the previous chapter. The following sections describe each of the functional areas that comprise the EBC.

CPU Selection

These four inputs to the EBC indicate the host CPU type and its bus frequency. Table 12-1 defines the valid settings for these inputs. If the host CPU is integrated onto the system board, these pins should be permanently strapped to the appropriate state. When the host CPU resides on a plug-in card, however, the four CPU signals should be set to the appropriate state when the CPU card is inserted. This allows automatic configuration of the EBC to match the CPU card installed in the machine. CPU input patterns not specified in table 12-1 are reserved for future use.

Table 12-1. CPU Type/Frequency

CPU3	CPU2	CPU1	CPU0	CPU Type/Frequency
1	0	1	0	32-bits, 2x clock, 25MHz 80386
1	0	1	1	32-bits, 2x clock, 33MHz 80386
1	1	0	0	32-bits, 1x clock, 25MHz 80486
1	1	0	1	32-bits, 1x clock, 33MHz 80486

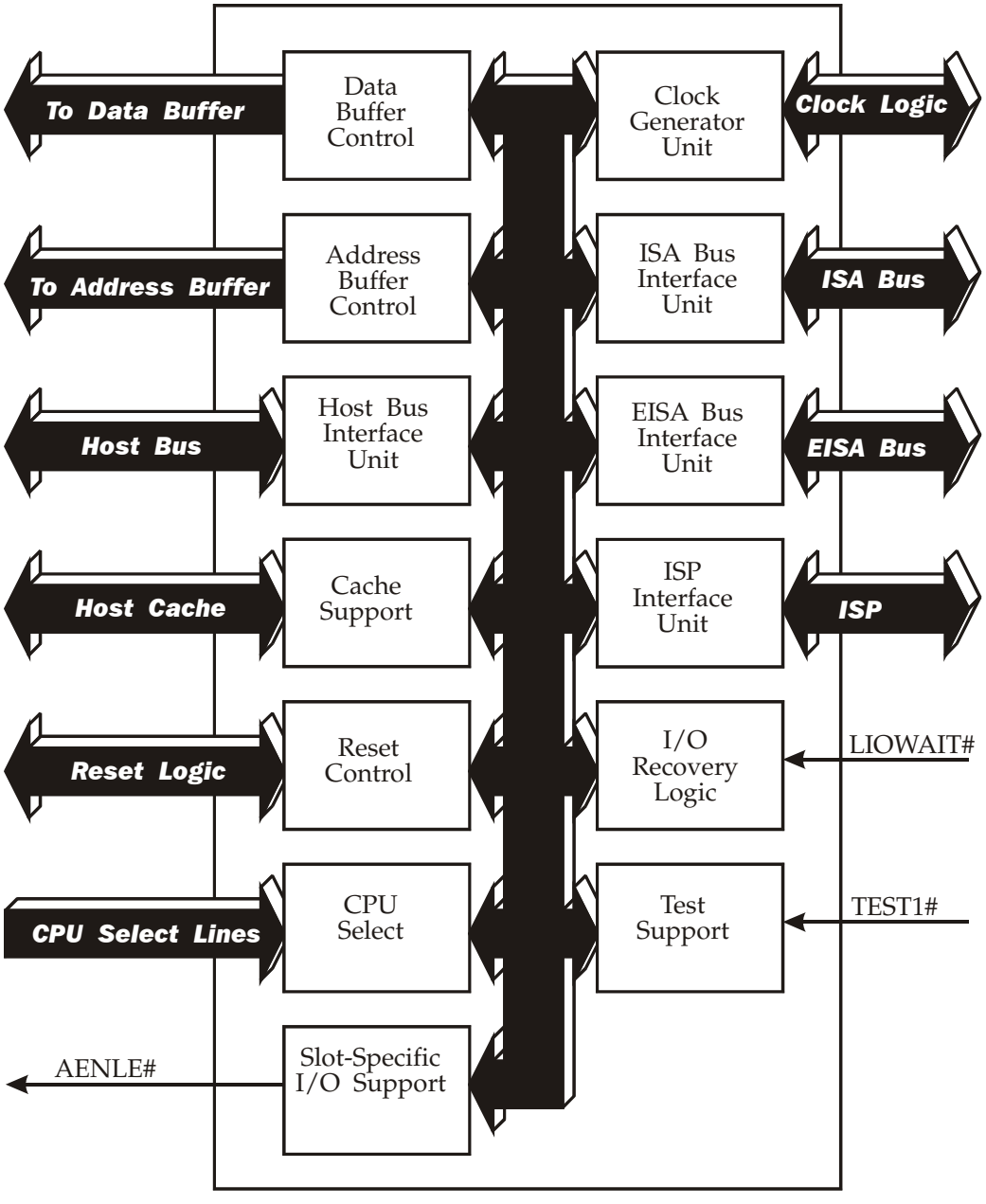


Figure 12-2. The Intel 82358DT EBC

Data Buffer Control and EISA Bus Buffer (EBB)

General

The EBC Data Buffer Control block pictured in figure 12-2 uses a group of EBC output signals to:

- control the data transceivers when routing data between the host and EISA buses.
- perform data bus steering when necessary, utilizing the latches and data bus transceivers.

These transceivers and latches are located in the 82352 EISA Bus Buffer, or EBB, pictured in figure 12-3. Table 12-2 defines the EBC output signals used to control the data EBB.

Table 12-2. EBC Output Signals Used to Control the Data EBB

Signal	Pin	Description
SDCPYEN01#	4	Enables the data EBB's steering transceiver between EISA data paths zero and one. The direction of copy is defined by the state of the SDCPYUP signal. If SDCPYUP is low, the byte on EISA data path one is copied to EISA data path zero. If SDCPYUP is high, the byte on EISA data path zero is copied to EISA data path one.
SDCPYEN02#	5	Enables the data EBB's steering transceiver between EISA data paths zero and two. The direction of copy is defined by the state of the SDCPYUP signal. If SDCPYUP is low, the byte on EISA data path two is copied to EISA data path zero. If SDCPYUP is high, the byte on EISA data path zero is copied to EISA data path two.
SDCPYEN03#	6	Enables the data EBB's steering transceiver between EISA data paths zero and three. The direction of copy is defined by the state of the SDCPYUP signal. If SDCPYUP is low, the byte on EISA data path three is copied to EISA data path zero. If SDCPYUP is high, the byte on EISA data path zero is copied to EISA data path three.

Table 12 - 2, cont.

Signal	Pin	Description
SDCPYEN13#	7	Enables the data EBB's steering transceiver between EISA data paths one and three. The direction of copy is defined by the state of the SDCPYUP signal. If SDCPYUP is low, the byte on EISA data path three is copied to EISA data path one. If SDCPYUP is high, the byte on EISA data path one is copied to EISA data path three.
SDCPYUP	8	See SDCPYEN01# description.
SDHDLE3#	10	When activated by the EBC, causes the data EBB to latch the data byte on EISA data path three.
SDHDLE2#	11	When activated by the EBC, causes the data EBB to latch the data byte on EISA data path two.
SDHDLE1#	12	When activated by the EBC, causes the data EBB to latch the data byte on EISA data path one.
SDHDLE0#	13	When activated by the EBC, causes the data EBB to latch the data byte on EISA data path zero.
SDOE2#	14	When activated by the EBC, causes the data EBB to drive the two previously latched bytes onto EISA data paths two and three.
SDOE1#	16	When activated by the EBC, causes the data EBB to drive the previously latched byte onto EISA data path one.
SDOE0#	17	When activated by the EBC, causes the data EBB to drive the previously latched byte onto EISA data path zero.
HDSDLE1#	18	When activated by the EBC, causes the data EBB to latch four bytes from the host data bus.
HDOE1#	20	When activated by the EBC, causes the data EBB to drive the two bytes latched into the path two and three latches onto paths two and three of the host data bus.
HDOE0	22	When activated by the EBC, causes the data EBB to drive the two bytes latched into the path zero and one latches onto paths zero and one of the host data bus.

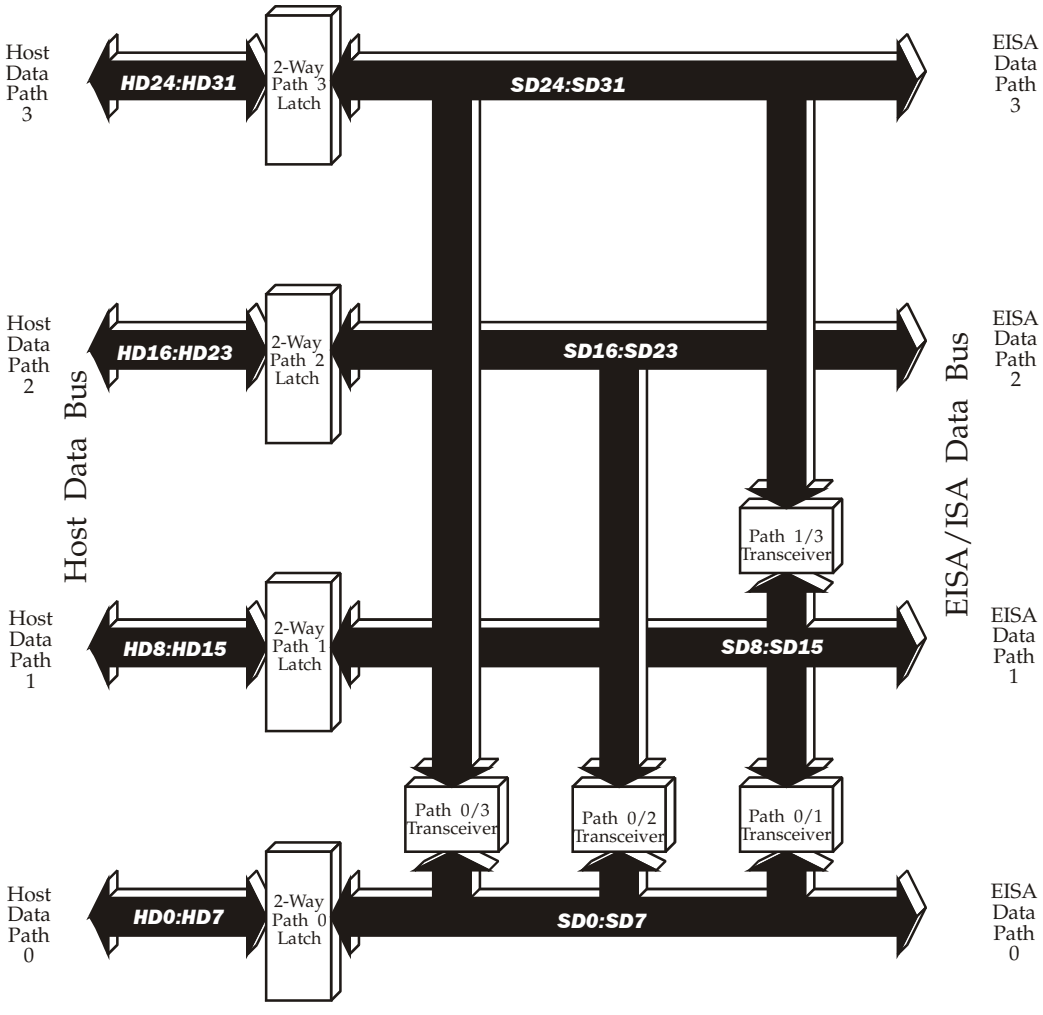


Figure 12-3. The Data EISA Bus Buffer, or EBB

Transfer Between 32-bit EISA Bus Master and 8-bit ISA Slave

Two examples are described in the following paragraphs: a 32-bit read from an 8-bit ISA slave; and a 32-bit write to an 8-bit ISA slave. Refer to figure 12-4 during the discussion.

In the first example, **the bus master is initiating a 32-bit read from an 8-bit ISA slave**. The 32-bit bus master begins the bus cycle by placing the double-

word address on LA[31:2], setting M/IO# to the appropriate state, and activating all four byte enable lines, BE#[3:0]. The bus master sets the W/R# bus cycle definition line low to indicate a read is in progress and activates the START# signal to indicate that the bus cycle has begun.

At the end of address time, which is one BCLK cycle in duration, the 32-bit bus master deactivates START#, the EBC activates CMD# and the bus master samples the EX32# line to see if a 32-bit EISA slave is responding. When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since an 8-bit ISA slave is being addressed in this example, EX32# is not sampled active by the bus master. At the end of address time, the EBC also samples EX32#, as well as EX16#, M16# and IO16# to determine the size and type of slave device that is responding. Since none of these four signals are sampled active, the EBC determines that the bus master is currently addressing an 8-bit ISA slave. Upon determining that the addressed slave is not connected to all four data paths, the bus master assumes that the EBC and EBB will take care of any data bus steering that may be necessary to accomplish the transfer. In order to let the EBC and EBB use the buses for steering, the bus master disconnects from the four data paths, the byte enable lines and the START# signal at midpoint of data time. The bus master continues to drive the doubleword address onto LA[31:2], however, as well as M/IO# and W/R#. The bus master then samples the state of the EX32# line at the end of each data time until it is sampled active. During this period of time, data bus steering is being performed by the EBC and EBB.

The EBC converts the M/IO# and W/R# settings to an active level on either the IORC#, SMRDC# or MRDC# bus cycle definition line on the ISA portion of the bus. The EBC also converts the active level on the byte enable lines to zeros on SA0 and SA1 and a low on SBHE#. The addressed 8-bit ISA slave responds to the read and drives the byte from the addressed location onto the lower data path, SD[7:0]. The EBC monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer and then latches the byte into the path zero latch in the data EBB using the EBC's SDHDLE0# output signal. The EBC deactivates CMD#.

Having completed the transfer of the first of the four bytes, the EBC increments the address by setting SA0 to a one, SA1 to a zero and SBHE# active. The EBC then tricks the addressed slave into thinking a new bus cycle has begun by generating START# again, followed by CMD#. When the EBC senses the changes on START# and CMD#, it turns the ISA command line off (SMRDC#, SMWTC#, IORC# or IOWC#) and then on again, causing the 8-bit ISA device to think another bus cycle has begun. The 8-bit ISA slave then drives the byte

from the currently addressed location onto data path zero, SD[7:0]. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The EBC then copies the byte to data path one and latches it into the data EBB's path one data latch. This is accomplished by activating the EBC's SDCPYEN01# and SDCPYUP output signals to copy the byte from path zero to path one and then latching the byte into the path one latch in the data EBB using the EBC's SDHDLE1# output signal. The first two of the four requested data bytes are now latched into the data EBB.

The EBC again increments the address by setting SA0 to a zero, SA1 to a one and SBHE# active. The EBC again tricks the addressed slave into thinking a new bus cycle has begun by generating START#, followed by CMD#, causing the appropriate ISA command line to be deactivated and then activated again. The 8-bit ISA slave then drives the byte from the currently addressed location onto data path zero, SD[7:0]. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The EBC then copies the byte to data path two and latches it into the data EBB's path two data latch. This is accomplished by activating the EBC's SDCPYEN02# and SDCPYUP output signals to copy the byte from path zero to path two and then latching the byte into the path two latch in the data EBB using the EBC's SDHDLE2# output signal. The first three of the four requested data bytes are now latched into the data EBB.

The EBC again increments the address by setting SA0 and SA1 high and SBHE# active. The EBC again tricks the addressed slave into thinking a new bus cycle has begun by generating START#, followed by CMD#, causing the appropriate ISA command line to be deactivated and then activated again. The 8-bit ISA slave then drives the byte from the currently addressed location onto data path zero, SD[7:0]. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The EBC then copies the byte to data path three and latches it into the data EBB's path three data latch. This is accomplished by activating the EBC's SDCPYEN03# and SDCPYUP output signals to copy the byte from path zero to path three and then latching the byte into the path three latch in the data EBB using the EBC's SDHDLE3# output signal. All four of the requested data bytes are now latched into the data EBB.

Using its SDOE0#, SDOE1# and SDOE2# outputs, the EBC now commands the data EBB to drive the four latched bytes onto the four data paths. The EBC activates the EX32# and EX16# lines at the midpoint of the current data time to signal the end of data bus steering. At the trailing-edge of the current data time, the 32-bit EISA bus master samples EX32# active, indicating that the necessary steering has been completed. The bus master can begin to drive the ad-

dress for the next bus cycle onto the buses at the midpoint of the next data time. The current bus cycle completes at the end of this last data time. Since this is a read bus cycle, the bus master reads the four bytes from the four data paths when the EBC deactivates CMD#, ending the bus cycle.

In the second example, **the bus master is initiating a 32-bit write to an 8-bit ISA slave**. The 32-bit bus master begins the bus cycle by placing the doubleword address on LA[31:2], setting M/IO# to the appropriate state, and activating all four byte enable lines, BE#[3:0]. The bus master sets the W/R# bus cycle definition line high to indicate a write is in progress and activates the START# signal to indicate that the bus cycle has begun. At the midpoint of address time, the bus master begins to drive the four bytes onto the four EISA data paths.

At the end of address time, which is one BCLK cycle in duration, the following events occur:

- The 32-bit bus master deactivates START# and the EBC activates CMD#.
- Using its four SDHDLEx# outputs, the EBC causes the data EBB to latch the four data bytes being driven onto the four EISA data paths by the bus master.
- The bus master samples the EX32# line to see if a 32-bit EISA slave is responding.

When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since an 8-bit ISA slave is being addressed in this example, EX32# is not sampled active by the bus master. At the end of address time, the EBC also samples EX32#, as well as EX16#, M16# and IO16# to determine the size and type of slave device that is responding. Since none of these four signals are sampled active, the EBC determines that the bus master is currently addressing an 8-bit ISA slave. Upon determining that the addressed slave is not connected to all four data paths, the bus master assumes that the EBC and EBB will take care of any data bus steering that may be necessary to accomplish the transfer. In order to let the EBC and EBB use the buses for steering, the bus master disconnects from the four data paths, the byte enable lines and the START# signal at midpoint of data time. The bus master continues to drive the doubleword address onto LA[31:2], however, as well as M/IO# and W/R#. The bus master then samples the state of the EX32# line at the end of each data time until it is sampled active. During this period of time, data bus steering is being performed by the EBC and EBB.

The EBC converts the M/IO# and W/R# settings to an active level on either the IOWC#, SMWTC# or MWTC# bus cycle definition line on the ISA portion of

the bus. The EBC also converts the active level on the byte enable lines to zeros on SA0 and SA1 and a low on SBHE#. Using its SDOE0# output, the EBC causes the data EBB to drive the byte latched in its path zero latch onto EISA data path zero. The addressed 8-bit ISA slave responds to the write and accepts the byte from the lower data path, SD[7:0]. The EBC monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The EBC de-activates CMD# and SDOE0#, causing the data EBB to cease driving the byte onto EISA data path zero.

Having completed the transfer of the first of the four bytes, the EBC increments the address by setting SA0 to a one, SA1 to a zero and SBHE# active. The EBC then tricks the addressed slave into thinking a new bus cycle has begun by generating START# again, followed by CMD#. This causes the appropriate ISA command line to be deactivated and then activated again. Using its SDCPYUP, SDCPYEN01# and SDOE1# output signals, the EBC causes the data EBB to drive the byte latched in its path one latch onto path one and copies it down to EISA data path zero. The 8-bit ISA slave then accepts the byte from EISA data path zero, SD[7:0]. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The first two of the four data bytes have been written to the target 8-bit ISA slave. The EBC deactivates CMD#, SDCPYEN01# and SDOE1#, causing the data EBB to cease driving the byte onto EISA data path one and turning off the EBB's copy transceiver.

The EBC again increments the address by setting SA0 to a zero, SA1 to a one and SBHE# active. The EBC again tricks the addressed slave into thinking a new bus cycle has begun by generating START#, followed by CMD#, causing the appropriate ISA command line to be deactivated and then activated again. Using its SDCPYUP, SDCPYEN02# and SDOE2# output signals, the EBC causes the data EBB to drive the byte latched in its path two latch onto path two and copies it down to EISA data path zero. The 8-bit ISA slave then accepts the byte from EISA data path zero, SD[7:0]. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The first three of the four data bytes have been written to the target 8-bit ISA slave. The EBC deactivates CMD#, SDCPYEN02# and SDOE2#, causing the data EBB to cease driving the byte onto EISA data path two and turning off the EBB's copy transceiver.

The EBC again increments the address by setting SA0 and SA1 high and SBHE# active. The EBC again tricks the addressed slave into thinking a new bus cycle has begun by generating START#, followed by CMD#, causing the appropriate ISA command line to be deactivated and then activated again. Using its SDCPYUP, SDCPYEN03# and SDOE2# output signals, the EBC causes

the data EBB to drive the byte latched in its path three latch onto path three and copies it down to EISA data path zero. The 8-bit ISA slave then accepts the byte from EISA data path zero, SD[7:0]. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. All four of the requested data bytes have now been written to the target 8-bit ISA slave. The EBC deactivates CMD#, SDCPYEN03# and SDOE2#, causing the data EBB to cease driving the byte onto EISA data path three and turning off the EBB's copy transceiver.

The EBC activates the EX32# and EX16# lines at the midpoint of the current data time to signal the end of data bus steering. At the trailing-edge of the current data time, the 32-bit EISA bus master samples EX32# active, indicating that the necessary steering has been completed. The bus master can begin to drive the address for the next bus cycle onto the buses at the midpoint of the next data time. The current bus cycle completes at the end of this last data time. Since this is a write bus cycle, the bus master ends the bus cycle when the EBC deactivates CMD#.

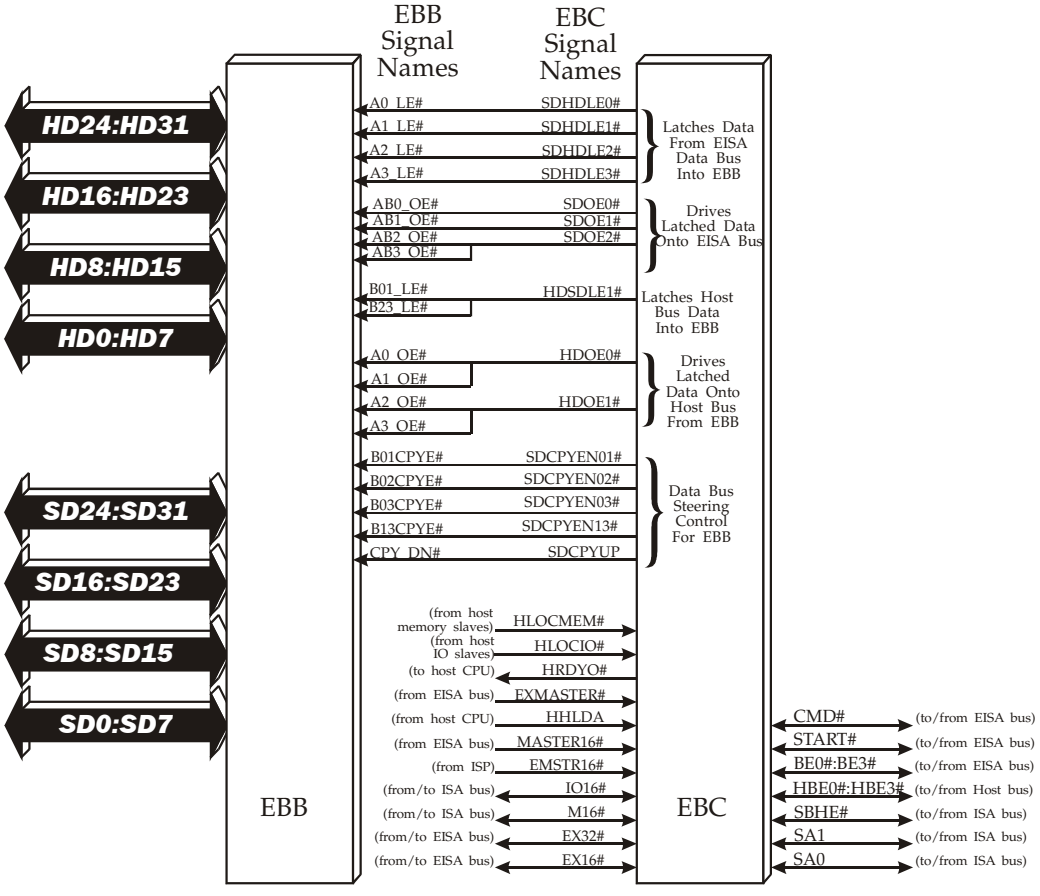


Figure 12-4. Linkage Between the EBC and the Data EBB

Transfer Between 32-bit EISA Bus Master and 16-bit ISA Slave

Two examples are described in the following paragraphs: a 32-bit read from the 16-bit ISA slave; and a 32-bit write to a 16-bit ISA slave. Refer to figure 12-4 during the discussion.

In the first example, **the bus master is initiating a 32-bit read from a 16-bit ISA slave**. The 32-bit bus master begins the bus cycle by placing the double-word address on LA[31:2], setting M/IO# to the appropriate state, and activating all four byte enable lines, BE#[3:0]. The bus master sets the W/R# bus cycle

definition line low to indicate a read is in progress and activates the START# signal to indicate that the bus cycle has begun.

The EBC determines that a 32-bit EISA bus master has initiated the bus cycle using the criteria in table 12-3. HHLDA, Host Hold Acknowledge, is inactive, indicating that the host CPU is not the bus master. EXMASTER# is active and MASTER16# is inactive, indicating that a 32-bit EISA bus master is using the buses.

Table 12-3. EBC's Bus Master Type Determination Criteria

HHLDA	REFRESH#	EXMASTER#	MASTER16#	EMSTR16#	MSBURST#	Bus Master Type
0	1	1	1	1	1	32-bit host CPU
1	0	1	1	x	1	Refresh
1	1	0	1	1	1	32-bit EISA
1	1	0	1	1	0	32-bit EISA burst
1	1	0	pulse	1	0	downshift 32-bit EISA burst
1	1	0	0	1	1	16-bit EISA
1	1	0	0	1	0	16-bit EISA burst
1	1	1	0	0	1	16-bit ISA
1	1	1	1	1	1	DMA
1	1	1	1	1	0	DMA burst

At the end of address time, which is one BCLK cycle in duration, the 32-bit bus master deactivates START#, the EBC activates CMD# and the bus master samples the EX32# line to see if a 32-bit EISA slave is responding. When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since a 16-bit ISA slave is being addressed in this example, EX32# is not sampled active by the bus master. At the end of address time, the EBC also samples EX32#, as well as EX16#, M16# and IO16# to determine the size and type of slave device that is responding. Since either M16# or IO16# is sampled active, the EBC determines that the bus master is currently addressing a 16-bit ISA slave. Upon determining that the addressed slave is not connected to all four data paths, the bus master assumes that the EBC and EBB will take care of any data bus steering that may be necessary to accomplish the transfer. In order to let the EBC and EBB use the buses for steering, the bus master disconnects from the four data paths, the byte enable lines and the START# signal at midpoint of data time. The bus master con-

tinues to drive the doubleword address onto LA[31:2], however, as well as M/IO# and W/R#. The bus master then samples the state of the EX32# line at the end of each data time until it is sampled active. During this period of time, data bus steering is being performed by the EBC and EBB.
The EBC converts the M/IO# and W/R# settings to an active level on either the IORC#, SMRDC# or MRDC# bus cycle definition line on the ISA portion of the bus. The EBC also converts the active level on the byte enable lines to zeros on SA0 and SA1 and a low on SBHE#. The low on SA0 and SBHE# indicates to the addressed 16-bit ISA slave that a 16-bit transfer is in progress. The addressed 16-bit ISA slave responds to the read and drives the byte from the even-addressed location onto the EISA data path zero, SD[7:0], and the byte from the odd-addressed location onto EISA data path one, SD[15:8]. The EBC monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer and then latches the two bytes into the path zero and path one latches in the data EBB using the EBC's SDHDLE0# and SDHDLE1# output signals. The EBC deactivates CMD#.

Having completed the transfer of the first two of the four bytes, the EBC increments the address by setting SA0 to a zero, SA1 to a one and SBHE# active. The EBC then tricks the addressed slave into thinking a new bus cycle has begun by generating START# again, followed by CMD#, causing the appropriate ISA command line to be deactivated and then activated again. The 16-bit ISA slave then drives the byte from the even-addressed location onto EISA data path zero, SD[7:0], and the byte from the odd-addressed location onto EISA data path one. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The EBC then copies the two bytes on paths zero and one to data paths two and three and latches them into the data EBB's path two and three data latches. This is accomplished by activating the EBC's SDCPYEN02#, SDCPYEN13# and SDCPYUP output signals to copy the byte from path zero to path two, the byte from path one to path three, and then latching the bytes into the path two and three latches in the data EBB using the EBC's SDHDLE2# and SDHDLE3# output signals. All four of the requested data bytes are now latched into the data EBB.

Using its SDOE0#, SDOE1# and SDOE2# outputs, the EBC now commands the data EBB to drive the four latched bytes onto the four data paths. The EBC activates the EX32# and EX16# lines at the midpoint of the current data time to signal the end of data bus steering. At the trailing-edge of the current data time, the 32-bit EISA bus master samples EX32# active, indicating that the necessary steering has been completed. The bus master can begin to drive the address for the next bus cycle onto the buses at the midpoint of the next data time. The current bus cycle completes at the end of this last data time. Since this

is a read bus cycle, the bus master reads the four bytes from the four data paths when the EBC deactivates CMD#, ending the bus cycle.

In the second example, **the bus master is initiating a 32-bit write to a 16-bit ISA slave**. The 32-bit bus master begins the bus cycle by placing the doubleword address on LA[31:2], setting M/IO# to the appropriate state, and activating all four byte enable lines, BE#[3:0]. The bus master sets the W/R# bus cycle definition line high to indicate a write is in progress and activates the START# signal to indicate that the bus cycle has begun. At the midpoint of address time, the bus master begins to drive the four bytes onto the four EISA data paths.

At the end of address time, which is one BCLK cycle in duration, the following events occur:

- The 32-bit bus master deactivates START# and the EBC activates CMD#.
- Using its four SDHDLEx# outputs, the EBC causes the data EBB to latch the four data bytes being driven onto the four EISA data paths by the bus master.
- The bus master samples the EX32# line to see if a 32-bit EISA slave is responding.

When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since a 16-bit ISA slave is being addressed in this example, EX32# is not sampled active by the bus master. At the end of address time, the EBC also samples EX32#, as well as EX16#, M16# and IO16# to determine the size and type of slave device that is responding. Since either M16# or IO16# is sampled active, the EBC determines that the bus master is currently addressing a 16-bit ISA slave. Upon determining that the addressed slave is not connected to all four data paths, the bus master assumes that the EBC and EBB will take care of any data bus steering that may be necessary to accomplish the transfer. In order to let the EBC and EBB use the buses for steering, the bus master disconnects from the four data paths, the byte enable lines and the START# signal at midpoint of data time. The bus master continues to drive the doubleword address onto LA[31:2], however, as well as M/IO# and W/R#. The bus master then samples the state of the EX32# line at the end of each data time until it is sampled active. During this period of time, data bus steering is being performed by the EBC and EBB.
The EBC converts the M/IO# and W/R# settings to an active level on either the IOWC#, SMWTC# or MWTC# bus cycle definition line on the ISA portion of the bus. The EBC also converts the active level on the byte enable lines to zeros on SA0 and SA1 and a low on SBHE#. Using its SDOE0# and SDOE1# outputs, the EBC causes the data EBB to drive the bytes latched in its path zero and one

latches onto EISA data paths zero and one. The addressed 16-bit ISA slave responds to the write and accepts the two bytes from the EISA data paths zero and one, SD[7:0] and SD[15:8]. The EBC monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The EBC deactivates CMD#, SDOE0# and SDOE1#, causing the data EBB to cease driving the two bytes onto EISA data paths zero and one.

The EBC increments the address by setting SA0 to a zero, SA1 to a one and SBHE# active. The EBC tricks the addressed slave into thinking a new bus cycle has begun by generating START#, followed by CMD#, causing the appropriate ISA command line to be deactivated and then activated again. Using its SDCPYUP, SDCPYEN02#, SDCPYEN13# and SDOE2# output signals, the EBC causes the data EBB to drive the two bytes latched in its path two and three latches onto paths two and three and copies them down to EISA data paths zero and one. The 16-bit ISA slave then accepts the two bytes from EISA data paths zero and one, SD[7:0] and SD[15:8]. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. All four of the four data bytes have been written to the target 16-bit ISA slave. The EBC deactivates CMD#, SDCPYEN02#, SDCPYEN13# and SDOE2#, causing the data EBB to cease driving the two bytes onto EISA data paths two and three and turning off the EBB's two copy transceivers.

The EBC activates the EX32# and EX16# lines at the midpoint of the current data time to signal the end of data bus steering. At the trailing-edge of the current data time, the 32-bit EISA bus master samples EX32# active, indicating that the necessary steering has been completed. The bus master can begin to drive the address for the next bus cycle onto the buses at the midpoint of the next data time. The current bus cycle completes at the end of this last data time. Since this is a write bus cycle, the bus master ends the bus cycle when the EBC deactivates CMD#.

Transfer Between 32-bit EISA Bus Master and 16-bit EISA Slave

Two examples are described in the following paragraphs: a 32-bit read from the 16-bit EISA slave; and a 32-bit write to a 16-bit EISA slave. Refer to figure 12-4 during the discussion.

In the first example, **the bus master is initiating a 32-bit read from a 16-bit EISA slave**. The 32-bit bus master begins the bus cycle by placing the doubleword address on LA[31:2], setting M/IO# to the appropriate state, and activating all four byte enable lines, BE#[3:0]. The bus master sets the W/R# bus cycle definition line low to indicate a read is in progress and activates the START# signal to indicate that the bus cycle has begun.

At the end of address time, which is one BCLK cycle in duration, the 32-bit bus master deactivates START#, the EBC activates CMD# and the bus master samples the EX32# line to see if a 32-bit EISA slave is responding. When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since a 16-bit EISA slave is being addressed in this example, EX32# is not sampled active by the bus master. At the end of address time, the EBC also samples EX32#, as well as EX16#, M16# and IO16# to determine the size and type of slave device that is responding. Since EX16# is sampled active, the EBC determines that the bus master is currently addressing a 16-bit EISA slave. Upon determining that the addressed slave is not connected to all four data paths, the bus master assumes that the EBC and EBB will take care of any data bus steering that may be necessary to accomplish the transfer. In order to let the EBC and EBB use the buses for steering, the bus master disconnects from the four data paths, the byte enable lines and the START# signal at midpoint of data time. The bus master continues to drive the doubleword address onto LA[31:2], however, as well as M/IO# and W/R#. The bus master then samples the state of the EX32# line at the end of each data time until it is sampled active. During this period of time, data bus steering is being performed by the EBC and EBB.

The active level on BE0# and BE1# indicates to the addressed 16-bit EISA slave that a 16-bit transfer is in progress involving the first two locations in the currently addressed doubleword. The addressed 16-bit EISA slave responds to the read and drives the byte from the even-addressed location onto the EISA data path zero, SD[7:0], and the byte from the odd-addressed location onto EISA data path one, SD[15:8]. The EBC monitors EXRDY to determine when the slave is ready to end the transfer and then latches the two bytes into the path

zero and path one latches in the data EBB using the EBC's SDHDLE0# and SDHDLE1# output signals. The EBC deactivates CMD#.

The EBC now addresses the last two bytes in the addressed doubleword by activating BE2# and BE3# and deactivating BE0# and BE1#. The EBC then tricks the addressed slave into thinking a new bus cycle has begun by generating START# again, followed by CMD#. The 16-bit EISA slave then drives the byte from the even-addressed location onto EISA data path zero, SD[7:0], and the byte from the odd-addressed location onto EISA data path one. The EBC again monitors EXRDY to determine when the slave is ready to end the transfer. The EBC then copies the two bytes on paths zero and one to data paths two and three and latches them into the data EBB's path two and three data latches. This is accomplished by activating the EBC's SDCPYEN02#, SDCPYEN13# and SDCPYUP output signals to copy the byte from path zero to path two, the byte from path one to path three, and then latching the bytes into the path two and three latches in the data EBB using the EBC's SDHDLE2# and SDHDLE3# output signals. All four of the requested data bytes are now latched into the data EBB.

Using its SDOE0#, SDOE1# and SDOE2# outputs, the EBC now commands the data EBB to drive the four latched bytes onto the four data paths. The EBC activates the EX32# and EX16# lines at the midpoint of the current data time to signal the end of data bus steering. At the trailing-edge of the current data time, the 32-bit EISA bus master samples EX32# active, indicating that the necessary steering has been completed. The bus master can begin to drive the address for the next bus cycle onto the buses at the midpoint of the next data time. The current bus cycle completes at the end of this last data time. Since this is a read bus cycle, the bus master reads the four bytes from the four data paths when the EBC deactivates CMD#, ending the bus cycle.

In the second example, **the bus master is initiating a 32-bit write to a 16-bit EISA slave**. The 32-bit EISA bus master begins the bus cycle by placing the doubleword address on LA[31:2], setting M/IO# to the appropriate state, and activating all four byte enable lines, BE#[3:0]. The bus master sets the W/R# bus cycle definition line high to indicate a write is in progress and activates the START# signal to indicate that the bus cycle has begun. At the midpoint of address time, the bus master begins to drive the four bytes onto the four EISA data paths.

At the end of address time, which is one BCLK cycle in duration, the following events occur:

- The 32-bit bus master deactivates START# and the EBC activates CMD#.

- Using its four SDHDLEx# outputs, the EBC causes the data EBB to latch the four data bytes being driven onto the four EISA data paths by the bus master.
- The bus master samples the EX32# line to see if a 32-bit EISA slave is responding.

When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since a 16-bit EISA slave is being addressed in this example, EX32# is not sampled active by the bus master. At the end of address time, the EBC also samples EX32#, as well as EX16#, M16# and IO16# to determine the size and type of slave device that is responding. Since EX16# is sampled active, the EBC determines that the bus master is currently addressing a 16-bit EISA slave. Upon determining that the addressed slave is not connected to all four data paths, the bus master assumes that the EBC and EBB will take care of any data bus steering that may be necessary to accomplish the transfer. In order to let the EBC and EBB use the buses for steering, the bus master disconnects from the four data paths, the byte enable lines and the START# signal at midpoint of data time. The bus master continues to drive the doubleword address onto LA[31:2], however, as well as M/IO# and W/R#. The bus master then samples the state of the EX32# line at the end of each data time until it is sampled active. During this period of time, data bus steering is being performed by the EBC and EBB.

The active level on BE0# and BE1# indicates to the addressed 16-bit EISA slave that a 16-bit transfer is in progress involving the first two locations in the currently addressed doubleword. Using its SDOE0# and SDOE1# outputs, the EBC causes the data EBB to drive the bytes latched in its path zero and one latches onto EISA data paths zero and one. The addressed 16-bit EISA slave responds to the write and accepts the two bytes from the EISA data paths zero and one, SD[7:0] and SD[15:8]. The EBC monitors EXRDY to determine when the slave is ready to end the transfer. The EBC deactivates CMD#, SDOE0# and SDOE1#, causing the data EBB to cease driving the two bytes onto EISA data paths zero and one.

The EBC now addresses the last two bytes in the addressed doubleword by activating BE2# and BE3# and deactivating BE0# and BE1#. The EBC tricks the addressed slave into thinking a new bus cycle has begun by generating START#, followed by CMD#. Using its SDCPYUP, SDCPYEN02#, SDCPYEN13# and SDOE2# output signals, the EBC causes the data EBB to drive the two bytes latched in its path two and three latches onto paths two and three and copies them down to EISA data paths zero and one. The 16-bit EISA slave then accepts the two bytes from EISA data paths zero and one, SD[7:0]

and SD[15:8]. The EBC again monitors EXRDY to determine when the slave is ready to end the transfer. All four of the four data bytes have been written to the target 16-bit EISA slave. The EBC deactivates CMD#, SDCPYEN02#, SDCPYEN13# and SDOE2#, causing the data EBB to cease driving the two bytes onto EISA data paths two and three and turning off the EBB's two copy transceivers.

The EBC activates the EX32# and EX16# lines at the midpoint of the current data time to signal the end of data bus steering. At the trailing-edge of the current data time, the 32-bit EISA bus master samples EX32# active, indicating that the necessary steering has been completed. The bus master can begin to drive the address for the next bus cycle onto the buses at the midpoint of the next data time. The current bus cycle completes at the end of this last data time. Since this is a write bus cycle, the bus master ends the bus cycle when the EBC deactivates CMD#.

Transfer Between 32-bit EISA Bus Master and 32-bit EISA Slave

Two examples are described in the following paragraphs: a 32-bit read from the 32-bit EISA slave; and a 32-bit write to a 32-bit EISA slave. Refer to figure 12-4 during the discussion.

In the first example, **the bus master is initiating a 32-bit read from a 32-bit EISA slave**. The 32-bit bus master begins the bus cycle by placing the double-word address on LA[31:2], setting M/IO# to the appropriate state, and activating all four byte enable lines, BE#[3:0]. The bus master sets the W/R# bus cycle definition line low to indicate a read is in progress and activates the START# signal to indicate that the bus cycle has begun.

At the end of address time, which is one BCLK cycle in duration, the 32-bit bus master deactivates START#, the EBC activates CMD# and the bus master samples the EX32# line to see if a 32-bit EISA slave is responding. When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since a 32-bit EISA slave is being addressed in this example, EX32# is sampled active by the bus master. At the end of address time, the EBC also samples EX32#, as well as EX16#, M16# and IO16# to determine the size and type of slave device that is responding. Since EX32# is sampled active, the EBC determines that the bus master is currently addressing a 32-bit EISA slave. Upon determining that the addressed slave is connected to all four data paths, the bus master recognizes that the EBC and EBB will not have to perform data bus steering.

The active level on all four byte enable lines indicates to the addressed 32-bit EISA slave that a 32-bit transfer is in progress involving all four locations in the currently addressed doubleword. The addressed 32-bit EISA slave responds to the read and drives the four bytes onto their respective EISA data paths. The bus master monitors EXRDY to determine when the slave is ready to end the transfer and latches the four bytes when the EBC deactivates CMD#.

In the second example, **the bus master is initiating a 32-bit write to a 32-bit EISA slave**. The 32-bit EISA bus master begins the bus cycle by placing the doubleword address on LA[31:2], setting M/IO# to the appropriate state, and activating all four byte enable lines, BE#[3:0]. The bus master sets the W/R# bus cycle definition line high to indicate a write is in progress and activates the START# signal to indicate that the bus cycle has begun. At the midpoint of address time, the bus master begins to drive the four bytes onto the four EISA data paths.

At the end of address time, which is one BCLK cycle in duration, the following events occur:

- The 32-bit bus master deactivates START# and the EBC activates CMD#.
- The bus master samples the EX32# line to see if a 32-bit EISA slave is responding.

When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since a 32-bit EISA slave is being addressed in this example, EX32# is sampled active by the bus master. At the end of address time, the EBC also samples EX32#, as well as EX16#, M16# and IO16# to determine the size and type of slave device that is responding. Since EX32# is sampled active, the EBC determines that the bus master is currently addressing a 32-bit EISA slave. Upon determining that the addressed slave is connected to all four data paths, the bus master recognizes that the EBC and EBB will not have to perform data bus steering.

The active level on the four byte enable lines indicates to the addressed 32-bit EISA slave that a 32-bit transfer is in progress involving all four locations in the currently addressed doubleword. The addressed 32-bit EISA slave responds to the write and accepts the four bytes from EISA data paths zero through three. The bus master monitors EXRDY to determine when the slave is ready to end the transfer. Since this is a write bus cycle, the bus master ends the bus cycle when the EBC deactivates CMD#.

Transfer Between 32-bit EISA Bus Master and 32-bit Host Slave

It should be noted that all host bus slaves are 32-bit devices. In this example, assume that a 32-bit EISA bus master initiates a bus cycle to read two bytes of data from an 32-bit host slave. Assume also that they are the first two bytes in the addressed doubleword. The 32-bit EISA bus master begins the bus cycle by placing the doubleword address on LA[31:2], setting M/IO# to the appropriate state, and activating byte enable lines BE0# and BE1#. The bus master sets the W/R# bus cycle definition line low to indicate a read is in progress and activates the START# signal to indicate that the bus cycle has begun.

The EBC determines that a 32-bit EISA bus master has initiated the bus cycle using the criteria in table 12-3. HHLDA, Host Hold Acknowledge, is inactive, indicating that the host CPU is not the bus master. EXMASTER# is active and MASTER16# is inactive, indicating that a 32-bit EISA bus master is using the buses. The EBC determines that a host slave is responding by sampling either HLOCMEM# or HLOCIO# active. Having already determined that the bus cycle was initiated by an EISA bus master, the EBC generates EX32# to inform the bus master that a 32-bit device is responding.

At the end of address time, which is one BCLK cycle in duration, the 32-bit bus master deactivates START#, the EBC activates CMD# and the bus master samples the EX32# line to see if a 32-bit EISA slave is responding. When the bus master is addressing an 8 or 16-bit ISA, a 16-bit EISA slave, or an 8 or 16-bit host slave, EX32# will not be returned active. Since a 32-bit slave is being addressed in this example, EX32# is sampled active by the bus master. Upon determining that the addressed slave is connected to all four data paths, the bus master recognizes that the EBC and EBB will not have to perform data bus steering.

The EBC propagates the state of the EISA byte enable lines through to the host byte enable lines, HBE#[3:0]. The active level on byte enable lines HBE0# and HBE1# indicates to the addressed 32-bit host slave that a 16-bit transfer is in progress involving the first two locations in the currently addressed doubleword. The addressed 32-bit host slave responds to the read and drives the two requested bytes onto their respective EISA data paths, HD[7:0] and HD[15:8]. The EBC causes the data EBB to latch the two bytes by activating its HDSDLE1# output. It then gates the two latched bytes onto paths zero and one of the EISA data bus by activating its SDOE0# and SDOE1# outputs.
The bus master monitors EXRDY to determine when the slave is ready to end the transfer and then latches the two bytes when the EBC deactivates CMD#.

Transfer Between 16-bit EISA Bus Master and 8-bit ISA Slave

This example assumes that a 16-bit EISA bus master is writing two bytes to the first two locations of a doubleword located within an 8-bit ISA slave. When the 16-bit EISA bus master initiates a bus cycle, it performs the following functions:

- Drives the MASTER16# line active to inform the EBC that a 16-bit bus master has initiated the bus cycle.
- Drives the doubleword address onto LA[31:2] and sets the M/IO# line to the appropriate state.
- Drives the START# signal active.
- sets W/R# and the byte enable lines to the appropriate states.
- During a write transfer, the bus master starts to drive data onto EISA data path zero and/or path one at the midpoint of address time.

At the trailing-edge of address time, the bus master deactivates START# and the EBC activates CMD# to indicate the beginning of data time. The bus master samples EX16# and EX32# to determine if the currently addressed device is attached to at least the lower two data paths. Since this example assumes that the bus master is addressing an 8-bit ISA slave, neither EX16# nor EX32# will be sampled active. Upon determining that the addressed slave is not connected to the lower two EISA data paths, the bus master assumes that the EBC and EBB will take care of any data bus steering that may be necessary to accomplish the transfer. Using its SDHDLE0# and SDHDLE1# outputs, the EBC causes the data EBB to latch the two bytes being driven onto EISA data paths zero and one by the bus master.

In order to let the EBC and EBB use the buses for steering, the bus master disconnects from the two data paths, the byte enable lines and the START# signal at the midpoint of data time. The bus master continues to drive the doubleword address onto LA[31:2], however, as well as M/IO# and W/R#. The bus master then samples the state of the EX16# line at the end of each data time until it is sampled active. During this period of time, data bus steering is being performed by the EBC and EBB.

When the EBC determines that an ISA device is responding, the EBC converts M/IO# and W/R# to an active level on one of the following ISA bus cycle definition signals:

- IORC#
- IOWC#
- MRDC#

- MWTC#
- SMRDC#
- SMWTC#

In this example, either the IOWC#, MWTC# or SMWTC# line would be activated by the EBC. The EBC also converts the setting on the EISA byte enable lines to the appropriate setting on SA0, SA1 and SBHE#. In this case, the active level on BE0# and BE1# would be converted to a low on SA0, SA1 and SBHE# on the ISA address bus, indicating that the bus master is addressing an even location and the next sequential odd location and will use the lower two data paths to transfer the two bytes. When the bus master has disconnected from the data bus, START# and the byte enable lines at the midpoint of data time, the EBC initiates the necessary data bus steering.

Using its SDOE0# output, the EBC causes the data byte latched into the data EBB's path zero latch to be driven onto path zero, SD[7:0]. This byte is written into the even-addressed location within the target 8-bit ISA slave. The EBC monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The EBC then deactivates CMD# and SDOE0#, causing the data EBB to cease driving the byte onto EISA data path zero.

Having completed the transfer of the first of the four bytes, the EBC increments the address by setting SA0 to a one, SA1 to a zero and SBHE# active. The EBC then tricks the addressed slave into thinking a new bus cycle has begun by generating START# again, followed by CMD#, causing the appropriate ISA command line to be deactivated and then activated again. Using its SDCPYUP, SDCPYEN01# and SDOE1# output signals, the EBC causes the data EBB to drive the byte latched in its path one latch onto path one and copies it down to EISA data path zero. The 8-bit ISA slave then accepts the byte from EISA data path zero, SD[7:0]. The EBC again monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. Both data bytes have now been written to the target 8-bit ISA slave. The EBC deactivates CMD#, SDCPYEN01# and SDOE1#, causing the data EBB to cease driving the byte onto EISA data path one and turning off the EBB's copy transceiver.

The EBC activates the EX32# and EX16# lines at the midpoint of the current data time to signal the end of data bus steering. At the trailing-edge of the current data time, the 16-bit EISA bus master samples EX16# active, indicating that the necessary steering has been completed. The bus master can begin to drive the address for the next bus cycle onto the buses at the midpoint of the next data time. The current bus cycle completes at the end of this last data time.

Since this is a write bus cycle, the bus master ends the bus cycle when the EBC deactivates CMD#.

Transfer Between 16-bit EISA Bus Master and 16-bit ISA Slave

This example assumes that a 16-bit EISA bus master is reading two bytes from the last two locations of a doubleword located within a 16-bit ISA slave. When the 16-bit EISA bus master initiates a bus cycle, it performs the following functions:

- drives the MASTER16# line active to inform the EBC that a 16-bit bus master has initiated the bus cycle.
- drives the doubleword address onto LA[31:2] and sets the M/IO# line to the appropriate state.
- drives the START# signal active.
- sets W/R# and the byte enable lines to the appropriate states. In this example, W/R# is set low, indicating a read, and BE2# and BE3# are set active.

At the trailing-edge of address time, the bus master deactivates START# and the EBC activates CMD# to indicate the beginning of data time. The bus master samples EX16# and EX32# to determine if the currently addressed device is at least attached to the lower two data paths and supports EISA bus cycle timing. Since this example assumes that the bus master is addressing a 16-bit ISA slave, neither EX16# nor EX32# will be sampled active. The EBC will, however, sample either M16# or IO16# active indicating a 16-bit ISA slave is responding. Upon determining that the addressed slave is not capable of responding to EISA bus cycle timing, the bus master assumes that the EBC and EBB will take care of any data bus steering that may be necessary to accomplish the transfer. In this particular example, a 16-bit EISA bus master is communicating with a 16-bit ISA slave. Since both devices are connected to EISA data paths zero and one, no steering is actually necessary. The bus master, however, having no indication as to whether the addressed ISA slave is an 8 or 16-bit device, assumes that steering may be necessary and surrenders the data bus, byte enable lines and START# to the EBC's control. This is done at the midpoint of data time. The bus master continues to drive the doubleword address onto LA[31:2], however, as well as M/IO# and W/R#. The bus master then samples the state of the EX16# line at the end of each data time until it is sampled active. During this period of time, data bus steering is being performed by the EBC and EBB.

When the EBC determines that an ISA device is responding, the EBC converts M/IO# and W/R# to an active level on one of the following ISA bus cycle definition signals:

- IORC#
- IOWC#
- MRDC#
- MWTC#
- SMRDC#
- SMWTC#

In this example, either the IORC#, MRDC# or SMRDC# line would be activated by the EBC. The EBC also converts the setting on the EISA byte enable lines to the appropriate setting on SA0, SA1 and SBHE#. In this case, the active level on BE2# and BE3# would be converted to a low on SA0, and SBHE# and a high on SA1 on the ISA address bus, indicating that the bus master is addressing an even location and the next sequential odd location and will use the lower two data paths to transfer the two bytes.

The 16-bit ISA device returns the two requested data bytes on EISA data paths zero and one and the EBC activates EX16# to inform the bus master that it may resume control of the bus cycle. The EBC monitors NOWS# and CHRDY to determine when the slave is ready to end the transfer. The EBC then deactivates CMD# and the bus master reads the two bytes from EISA data paths zero and one when CMD# goes inactive.

Transfer Between 16-bit EISA Bus Master and 16-bit EISA Slave

This example assumes that a 16-bit EISA bus master is reading two bytes from the last two locations of a doubleword located within a 16-bit EISA slave. When the 16-bit EISA bus master initiates a bus cycle, it performs the following functions:

- Drives the MASTER16# line active to inform the EBC that a 16-bit bus master has initiated the bus cycle.
- Drives the doubleword address onto LA[31:2] and sets the M/IO# line to the appropriate state.
- Drives the START# signal active.
- Drives W/R# and the byte enable lines to the appropriate states. In this example, W/R# is set low, indicating a read, and BE2# and BE3# are set active.

At the trailing-edge of address time, the bus master deactivates START# and the EBC activates CMD# to indicate the beginning of data time. The bus master samples EX16# and EX32# to determine if the currently addressed device is at least attached to the lower two data paths and supports EISA bus cycle timing. Since this example assumes that the bus master is addressing a 16-bit EISA slave, EX16# will be sampled active. The EBC will also sample EX16# active, indicating a 16-bit EISA slave is responding. Upon determining that the addressed slave is capable of responding to EISA bus cycle timing, the bus master assumes that no data bus steering will be necessary to accomplish the transfer. In this particular example, a 16-bit EISA bus master is communicating with a 16-bit EISA slave. Since both devices are connected to EISA data paths zero and one, no steering is necessary.

Using the active level on BE2# and BE3# to determine the requested bytes, the 16-bit EISA device returns the two requested data bytes on EISA data paths zero and one. The EBC monitors EXRDY to determine when the slave is ready to end the transfer. The EBC then deactivates CMD# and the bus master reads the two bytes from EISA data paths zero and one when CMD# goes inactive.

Transfer Between 16-bit EISA Bus Master and 32-bit EISA Slave

This example assumes that a 16-bit EISA bus master is reading two bytes from the last two locations of a doubleword located within a 32-bit EISA slave. When

the 16-bit EISA bus master initiates a bus cycle, it performs the following functions:

- drives the MASTER16# line active to inform the EBC that a 16-bit bus master has initiated the bus cycle.
- drives the doubleword address onto LA[31:2] and sets the M/IO# line to the appropriate state.
- drives the START# signal active.
- sets W/R# and the byte enable lines to the appropriate states. In this example, W/R# is set low, indicating a read, and BE2# and BE3# are set active.

At the trailing-edge of address time, the bus master deactivates START# and the EBC activates CMD# to indicate the beginning of data time. The bus master samples EX16# and EX32# to determine if the currently addressed device is at least attached to the lower two data paths and supports EISA bus cycle timing. Since this example assumes that the bus master is addressing a 32-bit EISA slave, EX32# will be sampled active. The EBC will also sample EX32# active, indicating a 32-bit EISA slave is responding. Upon determining that the addressed slave is capable of responding to EISA bus cycle timing, the bus master assumes that no data bus steering will be necessary to accomplish the transfer. In this particular example, a 16-bit EISA bus master is communicating with a 32-bit EISA slave. Since both devices are connected to EISA data paths zero and one, no steering is necessary.

Using the active level on BE2# and BE3# to determine the requested bytes, the 32-bit EISA device returns the two requested data bytes on EISA data paths two and three. Since the 16-bit EISA bus master expects to receive the two bytes back on EISA data paths zero and one, the EBC must command the data EBB to copy the two bytes from paths two and three to paths zero and one. This is accomplished by the EBC setting its SDCPYEN02# and SDCPYEN13# outputs active and its SDCPYUP output low.

The 16-bit EISA bus master monitors EXRDY to determine when the slave is ready to end the transfer. The EBC then deactivates CMD# and the bus master reads the two bytes from EISA data paths zero and one when CMD# goes inactive.

Transfer Between 16-bit ISA Bus Master and 8-bit ISA Slave

When the 16-bit ISA bus master initiates a bus cycle, the Central Arbitration Control in the ISP chip activates its EMSTR16# output to inform the EBC that a 16-bit ISA bus master is running a bus cycle. In addition, the ISA bus master sets MASTER16# active to indicate that it is a 16-bit bus master. The bus master places the address on SA[19:0], SBHE# and LA[23:17]. The EBC commands the address EBB to bridge this address over to the EISA address bus on LA[31:2] and the EBC converts SA0, SA1 and SBHE# to the correct setting on the EISA byte enable lines.

In this example, assume the 16-bit ISA bus master is performing a two byte write to an 8-bit ISA slave. The least significant bit of the address, SA0, would therefore be zero and SBHE# would be low to address the even address and the next sequential odd address as well. The bus master begins to drive the two bytes of data onto SD[7:0] and SD[15:8] halfway through address time and activates either the IOWC#, MWTC# or SMWTC# ISA bus cycle definition line during data time. The EBC and the 16-bit ISA bus master recognize that an 8-bit ISA slave is responding by sampling EX16#, EX32#, M16#, IO16#, HLOCMEM# and HLOCIO# inactive. Since there is no way to get an ISA bus master to temporarily float the data bus so the EBC and data EBB can perform the two necessary transfers, it is up to the ISA bus master to recognize that it is attempting to perform a 16-bit transfer with an 8-bit device and handle the multiple transfers itself.

The ISA bus master monitors NOWS# and CHRDY to determine when the 8-bit ISA slave is ready to end the transfer of the first byte over EISA data path zero. It then ceases to drive the first byte onto path zero and copies the second byte from path one, SD[15:8], to path zero, SD[7:0]. In addition, the ISA bus master increments the address by setting SA0 to a one and tricks the slave into thinking a second bus cycle has been initiated by momentarily turning off the write command line (IOWC#, MWTC# or SMWTC#) and then reactivating it. The slave accepts the second byte. The master once again monitors NOWS# and CHRDY to determine when the slave is ready to end the bus cycle. This completes the two byte transfer to the 8-bit ISA slave.

Transfer Between 16-bit ISA Bus Master and 16-bit ISA Slave

When the 16-bit ISA bus master initiates a bus cycle, the Central Arbitration Control in the ISP chip activates its EMSTR16# output to inform the EBC that a 16-bit ISA bus master is running a bus cycle. In addition, the ISA bus master sets MASTER16# active to indicate that it is a 16-bit bus master. The bus master

places the address on SA[19:0], SBHE# and LA[23:17]. The EBC commands the address EBB to bridge this address over to the EISA address bus on LA[31:2] and the EBC converts SA0, SA1 and SBHE# to the correct setting on the EISA byte enable lines.

In this example, assume the 16-bit ISA bus master is performing a two byte write to a 16-bit ISA slave. The least significant bit of the address, SA0, would therefore be zero and SBHE# would be low to address the even address and the next sequential odd address as well. The bus master begins to drive the two bytes of data onto SD[7:0] and SD[15:8] halfway through address time and activates either the IOWC#, MWTC# or SMWTC# ISA bus cycle definition line during data time. The EBC and the 16-bit ISA bus master recognize that a 16-bit ISA slave is responding when it samples EX16#, EX32#, M16#, IO16#, HLOCMEM# and HLOCIO# and senses either M16# or IO16# active.

The ISA bus master monitors NOWS# and CHRDY to determine when the 16-bit ISA slave is ready to end the transfer of the two bytes over EISA data paths zero and one. This completes the two byte transfer to the 16-bit ISA slave.

Transfer Between 16-bit ISA Bus Master and 16-bit EISA Slave

When the 16-bit ISA bus master initiates a bus cycle, the Central Arbitration Control in the ISP chip activates its EMSTR16# output to inform the EBC that a 16-bit ISA bus master is running a bus cycle. In addition, the ISA bus master sets MASTER16# active to indicate that it is a 16-bit bus master. The bus master places the address on SA[19:0], SBHE# and LA[23:17]. The EBC commands the address EBB to bridge this address over to the EISA address bus on LA[31:2] and the EBC converts SA0, SA1 and SBHE# to the correct setting on the EISA byte enable lines.

In this example, assume the 16-bit ISA bus master is performing a two byte write to a 16-bit EISA slave. The least significant bit of the address, SA0, would therefore be zero and SBHE# would be low to address the even address and the next sequential odd address as well. The EBC translates this to an active level on BE0# and BE1#. The EBC sets START# active during address time for the benefit of EISA slaves. The bus master begins to drive the two bytes of data onto SD[7:0] and SD[15:8] halfway through address time and activates either the IOWC#, MWTC# or SMWTC# ISA bus cycle definition line during data time. At the end of address time, the EBC sets CMD# active to indicate that it is data transfer time. The EBC converts the active ISA bus cycle line to the correct setting on the EISA bus cycle definition lines, M/IO# and W/R#. The EBC rec-

ognizes that a 16-bit EISA slave is responding when it samples EX16#, EX32#, M16#, IO16#, HLOCMEM# and HLOCIO# and senses EX16# active. If a memory bus cycle is in progress, the active level on EX16# is converted to an active level on M16#. If an I/O bus cycle is in progress, the active level on EX16# is converted to an active level on IO16#. This informs the ISA bus master that it is conversing with a 16-bit device and data bus steering is therefore unnecessary.

If the addressed EISA slave requires additional time to complete the transfer, it deactivates EXRDY until it is ready. The EBC converts EXRDY to CHRDY for the benefit of the ISA bus master. The ISA bus master monitors NOWS# and CHRDY to determine when the 16-bit EISA slave is ready to end the transfer of the two bytes over EISA data paths zero and one. This completes the two byte transfer to the 16-bit EISA slave.

Transfer Between 16-bit ISA Bus Master and 32-bit EISA Slave

In this example, assume that a 16-bit ISA bus master is writing two bytes of data to the second word of a doubleword within a 32-bit EISA slave. The bus master activates MASTER16# to inform the EBC that it is a 16-bit bus master. The Central Arbitration Control in the ISP chip activates EMSTR16# to inform the EBC that a 16-bit ISA bus master is performing a bus cycle.

The bus master places the address on SA[19:0], SBHE# and LA[23:17]. SA1 is set high, SA0 low and SBHE# low. The EBC activates START# during address time. EBC bridges this address across to the EISA address bus, LA[31:2], and converts SA0, SA1 and SBHE# to an active level on BE2# and BE3#. The EBC converts the active level on IOWC#, MWTC# or SMWTC# to the corresponding setting on the EISA bus cycle definition lines, M/IO# and W/R#. The EBC also deactivates START# and activates CMD# at the beginning of data transfer time. The bus master drives the two bytes onto EISA data paths zero and one.
Using its SDCPYEN02#, SDCPYEN13# and SDCPYUP outputs, the EBC causes the data EBB to copy the two bytes on paths zero and one to paths two and three. The addressed 32-bit EISA slave is expecting to receive the two bytes on the upper two data paths. The EBC monitors the EXRDY line to determine when the 32-bit EISA slave is ready to end the bus cycle. It then deactivates CMD#. The EISA slave latches the two data bytes from EISA data paths two and three when CMD# goes high at the end of the bus cycle.

Transfer Between 32-bit Host CPU and 32-bit Host Slave

All host bus I/O and memory devices are 32-bit devices. The EBC recognizes that the host CPU is performing a bus cycle when HHLDA, Host Hold Ac-

knowledge, is inactive and HADS0# and HADS1# are set active. The HADSx# lines are connected to the CPU's Host Address Status output. The host CPU places the address on HA[31:2] and sets the host byte enable lines, HBE#[3:0], to the appropriate state. The EBC causes the address EBB to broadcast the address onto the EISA and ISA address buses. The host CPU indicates the type of bus cycle on HM/IO#, HW/R# and HD/C#. When the host slave recognizes that it is being addressed, it activates either HLOCIO# (host local IO) or HLOCMEM# (host local memory).

Since the host CPU is communicating with a 32-bit slave on its own bus, the EBC and the data EBB do not become involved in the bus cycle. In other words, the data is not bridged over to the EISA/ISA bus.

Transfer Between 32-bit Host CPU and 8-bit ISA Slave

The EBC recognizes that the host CPU is performing a bus cycle when HHLDA, Host Hold Acknowledge, is inactive and HADS0# and HADS1# are set active. The HADSx# lines are connected to the CPU's Host Address Status output. The host CPU places the address on HA[31:2] and sets the host byte enable lines, HBE#[3:0], to the appropriate state. The EBC causes the address EBB to broadcast the address onto the ISA and EISA address buses as well. In this example, assume that the host CPU is writing two bytes to the 8-bit ISA slave over host data paths one and two. This means that the host CPU is setting BE1# and BE2# active. SA0 is set high, while SA1 and SBHE# are set low. The host CPU indicates the type of bus cycle on HM/IO#, HW/R# and HD/C#.

Since an 8-bit ISA slave is being addressed, the EBC samples inactive levels on M16#, IO16#, EX16#, EX32#, HLOCIO# and HLOCMEM#. The EBC latches the two bytes into the path one and two latches in the data EBB using its HDSDLE1# output. It then outputs the two bytes onto the EISA data bus by activating its SDOE1# and SDOE2# outputs. The data byte on EISA data path one is copied down to path zero when the EBC activates its SDCPYEN01# output and sets SDCPYUP low. The EBC monitors NOWS# and CHRDY to determine when the ISA slave is ready to end the byte transfer. The EBC turns off SDCPYEN01# and SDOE1# to turn off the copy transceiver and the cause the path one latch in the data EBB to stop outputting the first data byte.
Having completed the transfer of the first byte, the EBC then increments the address on the ISA address bus by setting SA1 and SBHE# high and SA0 low. The ISA slave is tricked into thinking another bus cycle is initiated by the EBC momentarily turning off the IOWC# or MWTC# line and then reactivating it. The EBC uses its SDCPYEN02# and SDCPYUP outputs to copy the second data byte from EISA data path two to path zero. The EBC again monitors NOWS#

and CHRDY to determine when the ISA slave is ready to end the byte transfer. The EBC turns off SDCPYEN02# and SDOE2# to turn off the copy transceiver and the cause the path two latch in the data EBB to stop outputting the second data byte.

Both bytes have now been transferred to the 8-bit ISA slave. The EBC now activates HRDYO#, host ready output, to tell the host CPU that it's ok to end the bus cycle.

Transfer Between 32-bit Host CPU and 16-bit ISA Slave

The EBC recognizes that the host CPU is performing a bus cycle when HHLDA, Host Hold Acknowledge, is inactive and HADS0# and HADS1# are set active. The HADSx# lines are connected to the CPU's Host Address Status output. The host CPU places the address on HA[31:2] and sets the host byte enable lines, HBE#[3:0], to the appropriate state. The EBC causes the address EBB to broadcast the address onto the ISA and EISA address buses as well. In this example, assume that the host CPU is writing two bytes to a 16-bit ISA slave over host data paths two and three. This means that the host CPU is setting BE2# and BE3# active. SA1 is set high, while SA0 and SBHE# are set low. The host CPU indicates the type of bus cycle on HM/IO#, HW/R# and HD/C#.

Since a 16-bit ISA slave is being addressed, the EBC samples an active level on M16# or IO16#. The EBC latches the two bytes into the path two and three latches in the data EBB using its HDSDLE1# output. It then outputs the two bytes onto the EISA data bus by activating its SDOE2# output. The data bytes on EISA data paths two and three are copied down to paths zero and one when the EBC activates its SDCPYEN02# and SDCPYEN13# outputs and sets SDCPYUP low. The EBC monitors NOWS# and CHRDY to determine when the ISA slave is ready to end the transfer. The EBC turns off SDCPYEN02#, SDCPYEN13# and SDOE2# to turn off the copy transceiver and the cause the path two and three latches in the data EBB to stop outputting the two data bytes.

Both bytes have now been transferred to the 16-bit ISA slave. The EBC now activates HRDYO#, host ready output, to tell the host CPU that it's ok to end the bus cycle.

Transfer Between 32-bit Host CPU and 16-bit EISA Slave

The EBC recognizes that the host CPU is performing a bus cycle when HHLDA, Host Hold Acknowledge, is inactive and HADS0# and HADS1# are

set active. The HADSx# lines are connected to the CPU's Host Address Status output. The host CPU places the address on HA[31:2] and sets the host byte enable lines, HBE#[3:0], to the appropriate state. The EBC causes the address EBB to broadcast the address onto the ISA and EISA address buses as well. In this example, assume that the host CPU is writing two bytes to a 16-bit EISA slave over host data paths two and three. This means that the host CPU is setting BE2# and BE3# active. The EBC activates BE2# and BE3# on the EISA address bus. The host CPU indicates the type of bus cycle on HM/IO#, HW/R# and HD/C#.

Since a 16-bit EISA slave is being addressed, the EBC samples an active level on EX16#. The EBC latches the two bytes into the path two and three latches in the data EBB using its HDSDLE1# output. It then outputs the two bytes onto the EISA data bus by activating its SDOE2# output. The data bytes on EISA data paths two and three are copied down to paths zero and one when the EBC activates its SDCPYEN02# and SDCPYEN13# outputs and sets SDCPYUP low. The EBC monitors EXRDY to determine when the EISA slave is ready to end the transfer. The EBC turns off SDCPYEN02#, SDCPYEN13# and SDOE2# to turn off the copy transceiver and the cause the path two and three latches in the data EBB to stop outputting the two data bytes.

Both bytes have now been transferred to the 16-bit EISA slave. The EBC now activates HRDYO#, host ready output, to tell the host CPU that it's ok to end the bus cycle.

Transfer Between 32-bit Host CPU and 32-bit EISA Slave

The EBC recognizes that the host CPU is performing a bus cycle when HHLDA, Host Hold Acknowledge, is inactive and HADS0# and HADS1# are set active. The HADSx# lines are connected to the CPU's Host Address Status output. The host CPU places the address on HA[31:2] and sets the host byte enable lines, HBE#[3:0], to the appropriate state. The EBC causes the address EBB to broadcast the address onto the ISA and EISA address buses as well. In this example, assume that the host CPU is writing four bytes to a 32-bit EISA slave using all four host data paths. This means that the host CPU is setting all four host byte enable lines, HBE#[3:0], active. The EBC activates BE#[3:0] on the EISA address bus. The host CPU indicates the type of bus cycle on HM/IO#, HW/R# and HD/C#.
Since a 32-bit EISA slave is being addressed, the EBC samples an active level on EX32#. The EBC latches the four bytes into the data EBB's data latches using its HDSDLE1# output. It then outputs the four bytes onto the EISA data bus by activating its SDOE0#, SDOE1# and SDOE2# outputs. The EBC monitors EXRDY

to determine when the EISA slave is ready to end the transfer. The EBC turns off SDOE0#, SDOE1# and SDOE2# to cause the four data EBB data latches to stop outputting the four data bytes.

All four bytes have now been transferred to the 32-bit EISA slave. The EBC now activates HRDYO#, host ready output, to tell the host CPU that it's ok to end the bus cycle.

Address Buffer Control and EBB

Under the control of the EBC, the address EBB ensures that the address generated by the current bus master is seen by every host, EISA and ISA slave in the system. Along with the address the state of the M/IO# bus cycle definition line must be propagated onto the EISA and host address buses so EISA and host slaves can discern memory addresses from I/O addresses. Table 12-4 defines the EBC output signals used to control the address EBB. Figure 12-5 provides a functional view of the address EBB and illustrates the linkage between the EBC and the address EBB. The figure also illustrates the direction of address flow through the three latching transceivers when the host CPU, an ISA master or an EISA master is the bus master. Table 12-5 shows the state of each of the EBC's address EBB control lines when each type of master is running a bus cycle. Entries designated as “transparent” indicate that the latch control line is left active for the entire bus cycle, causing the respective latching transceiver to be transparent.

Table 12-4. EBC Output Signals Used to Control the Address EBB

Signal	Description
HALAOE#	When set active by the EBC, causes the address EBB's upper and lower host/EISA latching transceivers to output the previously latched host address onto the EISA LA bus. LA[23:2] and LA#[31:24].
HALE#	When set active by the EBC, causes the address EBB's upper and lower host/EISA latching transceivers to latch the address on the EISA LA bus, LA[31:2].
LASAOE#	When set active by the EBC, causes the address EBB's EISA/ISA latching transceiver to output the previously LA address onto the SA bus, SA[19:2].
LAHAOE#	When set active by the EBC, causes the address EBB's upper and lower host/EISA latching transceivers to output the previously latched EISA address onto the host address bus, HA[31:2].
LALE#	When set active by the EBC, causes the address EBB's upper and lower host/EISA latching transceivers to latch the address on the host address bus, HA[31:2].
SALAOE#	When set active by the EBC, causes the address EBB's EISA/ISA latching transceiver to output the previously latched SA address onto LA bus, bits LA[16:2].
SALE#	When set active by the EBC, causes the address EBB to latch the address on LA[19:2] into the EISA/ISA latching transceiver.

Table 12-5. Address EBB Control Line States

	Current Bus Master Type				
Control Line	Host CPU	EISA	ISA	DMA	Refresh
HALAOE#	active	inactive	inactive	active	active
HALE#	transparent	transparent	transparent	transparent	transparent
LASAOE#	active	active	inactive	active	active
LAHAOE#	inactive	active	active	inactive	inactive
LALE#	pulsed to latch HA bus	na	na	transparent	transparent
SALAOE#	inactive	inactive	active	inactive	inactive
SALE#	pulsed to latch LA into SA latch	pulsed to latch LA into SA latch	transparent	pulsed to latch LA into SA latch	pulsed to latch LA into SA latch

Host CPU Bus Master

Refer to tables 12-5 and 12-4 and figure 12-5 while reading this description. When the host CPU is bus master, the address on the host address bus, HA[31:2], and the state of the HM/IO# bus cycle definition line must be propagated onto the EISA address bus, consisting of LA[23:2], LA#[31:24] and M/IO#, and the lower part of the ISA address bus, SA[19:2].

The pulse on LALE# causes the address EBB to latch the address from the host bus. The active on HALAOE# and the steady active on HALE# gates latched host address onto the EISA address bus, LA[23:2] and LA#[31:24]. It should be noted that the upper Host/EISA Latching Transceiver inverts address bits 31:24. The pulse on SALE# latches LA[19:2] into the EISA/ISA Latching Transceiver, while the active on LASAOE# allows it to output the latched address onto the SA bus, SA[19:2].

EISA Bus Master

Refer to tables 12-5 and 12-4 and figure 12-5 while reading this description. When an EISA master is the bus master, the address on the EISA address bus, LA[23:2] and LA#[31:24], and the state of the M/IO# bus cycle definition line must be propagated onto the host address bus, consisting of HA[31:2] and HM/IO#, and onto the lower part of the ISA address bus, SA[19:2].

The active on LAHAOE# and the steady active on HALE# allows the address on the EISA address bus to flow onto the host address bus. The pulse on SALE# causes the lower part of the EISA address, LA[19:2], to be latched into the EISA/ISA latching transceiver, while the active on LASAOE# allows the latched LA address to be driven onto the SA bus, SA[19:2].

ISA Bus Master

Refer to tables 12-5 and 12-4 and figure 12-5 while reading this description. When an ISA master is the bus master, the address on the ISA address bus, SA[19:2] and LA[23:17], must be propagated onto the host address bus, HA[31:2], and onto the lower part of the EISA address bus, LA[23:2]. Since ISA bus masters do not use LA#[31:24], pull-up resistors force these lines inactive when an ISA bus master is placing an address on the address bus.

The steady active state of SALE# and the active state of SALAOE# allows the portion of the ISA address on SA[16:2] to flow through the EISA/ISA latching transceiver onto the lower part of the EISA address bus, LA[16:2]. The ISA bus

master places address bits 23:17 directly onto LA[23:17] of the EISA/ISA address bus. The active on LAHAOE# and the steady active state of HALE# permits the address on the EISA address bus, LA[23:2] and LA#[31:24], to flow through onto the host address bus, HA[31:2]. The ones on LA#[31:24] are inverted by the upper Host/EISA latching transceiver before being driven onto HA[31:2]4.

Refresh Bus Master

Refer to tables 12-5 and 12-4 and figure 12-5 while reading this description. The Refresh logic is located in the ISP chip. When the Refresh logic becomes bus master and drives the next sequential row address onto the host address bus, the row address must be propagated onto the EISA and ISA addresses buses as well.

The active state of HALAOE# and the steady active state of LALE# allows the row address to flow from the host address bus, HA[31:2], to the EISA address bus, LA[31:2]. The pulse on SALE# latches the row address into the EISA/ISA latching transceiver and the active state of LASAOE# causes the row address to be driven onto the SA bus, SA[19:2]. The Refresh logic in the ISP also sets the HM/IO# bus cycle definition line high to indicate that a memory row address is on the bus. The EBC passes the state of the host bus HM/IO# line to the EISA M/IO# line.

DMA Bus Master

Refer to tables 12-5 and 12-4 and figure 12-5 while reading this description. The DMA controllers are located in the ISP chip and output a memory address onto the host address bus, HA[31:2], when a DMA channel becomes bus master. The HM/IO# line is also set high by the ISP to indicate that a memory address is present on the bus. The EBC must command the address EBB to pass the memory address and the state of HM/IO# onto the EISA address bus, LA[23:2] and LA#[31:24] plus M/IO#, and onto the ISA address bus, consisting of SA[19:2] and LA[23:17].

The active state of HALAOE# and the steady active state of LALE# allows the memory address on the host address bus, HA[31:2], to flow through the upper and lower Host/EISA latching transceivers onto the EISA data bus, LA[23:2] and LA#[31:24]. The pulse on SALE# latches the memory address on the host address bus, HA[31:2], into the EISA/ISA latching transceiver and the active state of LASAOE# allows the transceiver to drive it onto the SA bus, SA[19:2].

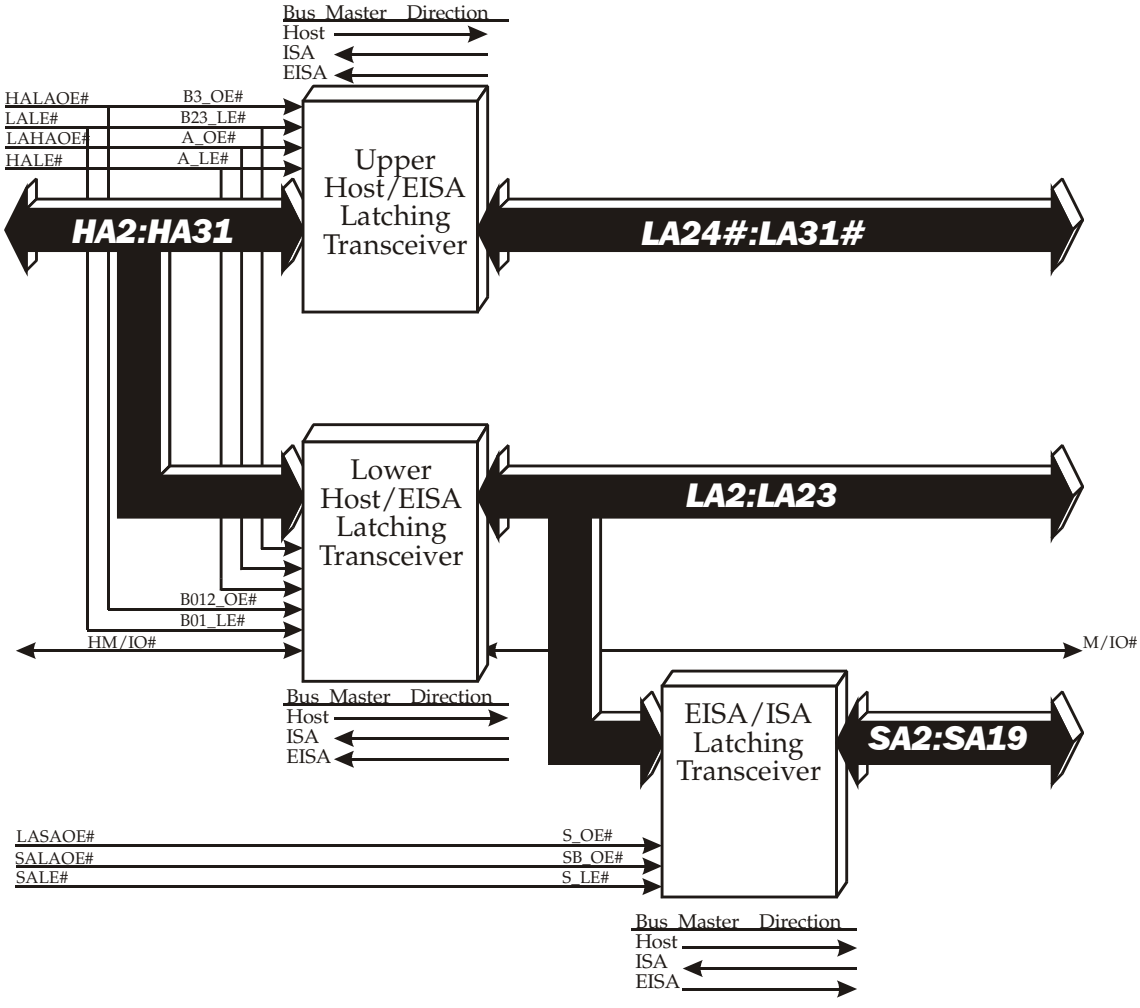


Figure 12-5. Block Diagram of Address EBB

Host Bus Interface Unit

The host bus interface unit pictured in figure 12-2 observes bus cycles initiated by the host CPU. If neither HLOCMEM# nor HLOCIO# are sensed active, the host bus master is addressing a slave on the EISA or ISA bus. In this case, the host bus interface unit commands either the EISA or ISA interface unit in the EBC to run a bus cycle. The host bus interface unit awaits completion of the bus

cycle and sends ready to the host CPU. Table 12-6 provides a description of the host bus interface signals. The description of these signals assumes that the EBC is configured for the 82350 environment. To configure the EBC for the 82350 environment, two conditions must be met:

- The AMODE input must be strapped low.
- The HNA#/SBMODE# input is sampled on the leading-edge of the SPWROK input. To select the 82350 configuration, it must be sampled high.

Table 12-6. Host Interface Unit Signal Descriptions

Signal	Direction	Description
AMODE	input	**Address Mode**. Configures the EBC for 82350 mode when strapped low; for 82350DT mode when strapped high.
HBE#[3:0]	input/output	**Host Byte Enables**. When the host CPU is bus master, these inputs define the target location(s) within the addressed doubleword. The EBC's ISA interface unit converts them to SA0, SA1 and SBHE# on the ISA address bus, while the EBC's EISA interface unit converts them to BE#[3:0] on the EISA address bus. When an EISA bus master has initiated a bus cycle, the state of the BE#[3:0] lines on the EISA address bus are output onto the HBE#[3:0] lines on the host address bus. When an ISA bus master has initiated a bus cycle, the state of the SA0, SA1 and SBHE# lines on the ISA address bus are converted and output onto the HBE#[3:0] lines on the host address bus.
HADS0# and HADS1#	input	**Host Address Status 0 and 1**. The host CPU or the host cache controller's ADS# output is connected to the HADS0# input. ADS# indicates that it is address time and a valid address and bus cycle definition are present on the host bus. Some cache controllers perform more than one fetch in order to fill a cache line. In this case, the cache controller generates HADS0# when it initiates the bus cycle for the first fetch. This triggers an state machine that generates HADS1# when it initiates any subsequent bus cycles for the remaining fetches. Internally, these two input signals are “anded” together.

Table 12-6, cont.

HNA#	output	**Host Next Address**. In a system with a 386 host CPU, this output is used to tell the 386 whether it can output the address for the next bus cycle early.
HD/C#	input/output	**Host Data or Control**. Used as inputs when the host CPU is bus master, as outputs when a device other than the host CPU is bus master. In combination with HW/R# and HM/IO#, defines the bus cycle type.
HW/R#	input/output	**Host Write or Read**. See HD/C#.
HM/IO#	input/output	**Host Memory or I/O.** See HD/C#.
HLOCK#	input	**Host Lock**. This input is connected to the host CPU's LOCK# output. Will be active when the host CPU is locking multiple bus cycles together to prevent other bus masters from requesting the buses until lock goes inactive.
HRDYI#	input	**Host Ready Input.** The host interface unit monitors this signal to determine when a host-initiated bus cycle has completed.
HRDYO#	output	**Host Ready Output**. When the host CPU is accessing an EISA or ISA slave, the host interface unit activates HRDYO# to signal the end of the bus cycle to the host CPU.
HERDYO#	output	**Host Early Ready Output**. This is an earlier version of HRDYO# to be used with higher speed host CPUs that require more setup time.
HHOLD	output	**Host Hold Request**. When the Central Arbitration Control in the ISP chip must grant the buses to a device other than the host CPU, it must first take the buses away from the host CPU. To do this, the ISP activates DHOLD. DHOLD causes the EBC's host interface unit, in turn, to activate HHOLD to seize the host bus from the host CPU. In response, the host CPU surrenders the buses and activates HHLDA, Host Hold Acknowledge. The EBC then activates DHLDA to inform the Central Arbitration Control in the ISP that it may grant the buses to another device.
HHLDA	input	**Host Hold Acknowledge**. See HHOLD.

HLOCMEM#	input	**Host Local Memory**. This signal is set active by the memory address decode logic when memory residing on the host bus is being addressed. If the current bus master is the host CPU, this means that the EBC does not have to activate the data EBB or run a bus cycle on the ISA or EISA bus.
HLOCIO#	input	**Host Local I/O.** This signal is set active by the I/O address decode logic when an I/O device residing on the host bus is being addressed. If the current bus master is the host CPU, this means that the EBC does not have to activate the data EBB or run a bus cycle on the ISA or EISA bus.
HGT16M#	input	**Host Greater Than 16MB**. This signal is only driven by the ISP chip during DMA bus cycles. If the DMA channel is generating a memory address below 16MB (00000000h – 00FFFFFFh), HGT16M# is high and the ISA interface unit will generate MRDC# or MWTC#. For addresses above 16MB, the MRDC# or MWTC# signals are not generated. This is necessary because some DMA devices use the ISA memory command signals to start a bus cycle early.
PWEN#	input	**Posted Write Enable**. If sampled active at the beginning of a host CPU memory write bus cycle to an EISA or ISA memory slave, the EBC's host interface unit causes the EBC's Data Buffer Control logic to latch the write data into the data EBB. The host interface unit then activates the HRDYO# signal to let the host CPU end the memory write bus cycle. The EBC's host interface unit, in conjunction with either the ISA or EISA interface unit, then writes the posted data to the target ISA or EISA memory slave. This feature allows single host memory writes to EISA or ISA memory to complete quickly.

HSTRETCH#	input	**Host Bus Stretch.** This input can be used by host bus slaves during EISA/ISA or DMA bus master cycles to stretch the low part of BCLK during CMD# (data time). This has the effect of stalling the EISA/ISA master without adding BCLK wait states.
HKEN#	input	**Host Cache Enable**. When sampled active, indicates that the host CPU is requesting a cache line fill operation.

ISA Bus Interface Unit

The ISA interface unit pictured in figure 12-2 observes bus cycles initiated by ISA bus masters. The ISA bus interface unit awaits completion of the bus cycle. If either the host CPU or an EISA bus master is addressing an ISA slave, the ISA interface unit runs a bus cycle. When the bus cycle on the ISA bus is completed, EXRDY or HRDYO# is sent to the EISA or host bus master to terminate the bus cycle. Table 12-7 provides a description of the ISA interface signals. For a complete description of the ISA bus, refer to the MindShare book entitled *ISA System Architecture*.

Table 12-7. ISA Interface Unit Signal Descriptions

Signal	Direction	Description
BALE	output	**Bus Address Latch Enable**. During an ISA bus cycle, BALE is set high at the midpoint of address time and dropped low at the end of address time. The address is gated from the LA bus to the SA bus when BALE goes high and is latch when BALE goes low at the end of address time.
SA0, SA1, BHE#	input/output	Least-significant part of the ISA address bus. These are inputs when the bus cycle is being run by an ISA bus master and outputs when the bus cycle is being run by an EISA or host master.
IORC#	input/output	The **I/O Read Command line**. Generated by an ISA bus master when it is performing an I/O read bus cycle. When an EISA or host bus master is performing an I/O read bus cycle, the EBC's ISA interface unit generates this signal.
IOWC#	input/output	The **I/O Write Command line**. Generated by an ISA bus master when it is performing an I/O write bus cycle. When an EISA or host bus master is performing an I/O write bus cycle, the EBC's ISA interface unit generates this signal.
MRDC#	input/output	The **Memory Read Command line**. Generated by an ISA bus master when it is performing a memory read bus cycle. When an EISA or host bus master is performing a memory read bus cycle, the EBC's ISA interface unit generates this signal.
MWTC#	input/output	The **Memory Write Command line**. Generated by an ISA bus master when it is performing a memory write bus cycle. When an EISA or host bus master is performing a memory write bus cycle, the EBC's ISA interface unit generates this signal.
SMRDC#	output	**Standard Memory Read Command line**. The EBC's ISA interface unit generates this signal when any bus master is reading from memory space in the 00000000h – 000FFFFFh range. A memory address decoder located in the ISP chip generates GT1M# whenever it detects a memory address in this range, causing the ISP interface unit to generate either SMRDC# or SMWTC#.

Table 12 - 7, cont.

SMWTC#	output	**Standard Memory Write Command line**. The EBC's ISA interface unit generates this signal when any bus master is writing to memory space in the 00000000h – 000FFFFFh range. A memory address decoder located in the ISP chip generates GT1M# whenever it detects a memory address in this range, causing the ISP interface unit to generate either SMRDC# or SMWTC#.
IO16#	input/output	**IO Size 16**. Generated by a 16-bit ISA I/O slave when addressed by a bus master. Set active by the EBC's ISA interface unit when an ISA bus master is addressing a host I/O slave (HLOCIO# sampled active). EISA slaves that support ISA bus masters must assert IO16# as well as EX16# or EX32# when addressed.
M16#	input	**Memory Size 16**. Generated by a 16-bit ISA memory slave when addressed by a bus master.
NOWS#	input	**No Wait States**. An ISA slave may generate NOWS# when it has decoded its address and a read or write command line has been activated. When set active by the slave, it conditions the default ready timer (in the ISA interface unit) to set ready active at the end of the current BCLK. It is used to shorten the number of wait states appended to a bus cycle by the default ready timer.
CHRDY	input/output	**Channel Ready**. If an ISA slave requires more time to complete a bus cycle than allowed by the default ready timer, it may set the CHRDY line low. This prevents the default ready timer from timing out until the slave is ready to end the bus cycle. When the slave is ready to end the bus cycle, it sets CHRDY active again, permitting the default ready timer to time out.
REFRESH#	input	Generated by the Refresh logic in the ISP chip when the Refresh logic is bus master and is performing a refresh bus cycle.
MASTER16#	input	**16-bit Bus Master**. Generated by either ISA or 16-bit EISA bus master when it initiates a bus cycle.

EISA Bus Interface Unit

The EISA interface unit pictured in figure 12-2 observes bus cycles initiated by EISA bus masters. The EISA bus interface unit awaits completion of the bus cycle. If either the host CPU or an ISA bus master is addressing an EISA slave, the EISA interface unit runs a bus cycle. Table 12-8 provides a description of the EISA interface signals. For a complete description of the EISA bus, refer to earlier sections of this publication.

Table 12-8. EISA Interface Unit Signal Descriptions

Signal	Direction	Description
BE#[3:0]	input/output	**Byte Enable lines**. Set to the appropriate states during a bus cycle initiated by an EISA bus master. When an ISA master initiates a bus cycle, the EBC's EISA interface unit converts SA0, SA1 and SBHE# to the corresponding setting on the BE lines. When the host CPU initiates a bus cycle, the state of the host byte enables lines, HBE#[3:0], are passed onto the EISA byte enable lines by the EBC.
M/IO#	input/output	**Memory or I/O** bus cycle definition line. Generated by an EISA master or by the EBC when the host CPU or an ISA master is performing a bus cycle.
W/R#	input/output	**Write or Read** bus cycle definition line. Generated by an EISA master or by the EBC when the host CPU or an ISA master is performing a bus cycle.
LOCK#	output	Generated by the EBC when the host CPU is bus master and has asserted HLOCK# to the EBC. An active level on the EISA LOCK# line prevents other bus masters from requesting the buses until LOCK# goes away.
START#	input/output	Set active by an EISA bus master when it initiates a bus cycle and deactivated at the end of address time. Controlled by the EBC when an ISA master or the host CPU is performing a bus cycle to signal the start of the bus cycle on the EISA bus. Also controlled by the EBC during DMA transfers and under some conditions requiring data bus steering.
CMD#	output	**Command**. Set active by the EBC at the start of data time and kept active until the end of the bus cycle.

Table 12 - 8, cont.

EXRDY	input/output	**EISA Ready**. Set inactive by the currently addressed EISA slave if it requires more time to complete a bus cycle. Controlled by the EBC when an EISA bus master is addressing an ISA or host slave and under some conditions when data bus steering is necessary.
MSBURST#	input/output	**Master Burst.** Generated by an EISA or host master (through the EBC) when the addressed slave has indicated it supports burst bus cycles by asserting SLBURST#. Set active by the EISA master or the EBC at the midpoint of the first data time and sampled by the slave at the end of the first data time.
SLBURST#	input	**Slave Burst**. Used by the currently addressed EISA slave to indicate it supports burst bus cycles. Sampled by the master at the end of address time.
EX32#	input/output	**EISA Size 32**. Generated by the currently addressed slave if it is a 32-bit EISA slave. Generated by the EBC at the end of data bus steering to signal return of bus cycle control to the EISA bus master. Also generated by the EBC if an EISA bus master addresses a host slave (HLOCMEM# or HLOCIO# sampled active be EBC).
EX16#	input/output	**EISA Size 16**. Generated by the currently addressed slave if it is a 16-bit EISA slave. Generated by the EBC at the end of data bus steering to signal return of bus cycle control to the EISA bus master. Also generated by the EBC if an EISA bus master addresses a host slave (HLOCMEM# or HLOCIO# sampled active be EBC).

Cache Support

The EBC provides two output signals to support bus snooping. HSSTRB# is used in 386/82385 host CPU systems, while QHSSTRB# is used in 486 host CPU systems. These signals indicate to a system cache controller that a bus master is writing to system memory. The RDE#, or Ready Delay Enable, input instructs the cache support unit in the EBC to add a wait state by delaying HERDYO# and HRDYO# during a host to EISA/ISA read to allow increased cache SRAM write data setup time during cache read miss bus cycles.

Reset Control

Table 12-9 describes the signals associated with the EBC's Reset Control unit (see figure 12-2).

Table 12-9. The EBC's Reset Control Interface Signals

Signal	Direction	Description
RST	out	**System Reset**. Remains active until 10 microseconds after SPWROK goes active. Resets major system components to a known state.
RSTCPU	out	**Reset Host CPU**. Driven active when: power isn't stable (SPWROK inactive); a shutdown bus cycle is detected on the host bus; or RSTAR# is sensed active. See RSTAR# description. RSTCPU resets just the host CPU.
RST385	out	**Reset 385 cache controller**. Driven active under the same conditions as RSTCPU. Resets the host bus cache controller, clearing all tag valid bits. In other words, the cache is flushed.
RSTAR#	in	**Restart**. Generated under software control by: issuing a CPU reset command to the keyboard controller; or toggling the fast hot reset bit in the PS/2 compatibility port at I/O port 92h. Used by 286-specific code to reset the 286 and return it to real mode from protected mode.
SPWROK	in	**System Power OK**. Provided as an output from the power supply. When inactive, the Reset Control unit sets RSTCPU, RST385 and RST active.

Slot-Specific I/O Support

During I/O bus cycles, the EBC generates a pulse on the signal AENLE#, AEN Latch Enable. This is used by the AEN logic to latch the active AENx line. For more information on slot-specific I/O support and AEN decode, refer to the chapter entitled "EISA System Configuration."

Clock Generator Unit

Table 12-10 provides a description of the signals related to the EBC's Clock Generator unit.

Table 12-10. EBC's Clock Generation Unit Signal Description

Signals	Direction	Description
HCLKCPU	in	**Host CPU Clock**.
BCLK	out	**Bus Clock**. BCLK is generated by dividing the HCLKCPU input clock by a factor determined by the CPU[3:0] inputs. For a 25MHz 386 or 486 host CPU, the BCLK frequency will be 8.33MHz. For a 33MHz 386 or 486, the BCLK frequency will be 8.25MHz.
CLKKB	out	**Keyboard Clock**. The EBC's Clock Generation unit provides an output clock for the keyboard controller. Its frequency is derived by dividing the HCLKCPU input by a factor determined by the CPU[3:0] inputs. If the host CPU is a 25MHz 386, the CLKKB output frequency will be 10MHz. If the host CPU is a 25MHz 486, the CLKKB output frequency will be 8.33MHz. If the host CPU is a 33MHz 386 or 486, the CLKKB output frequency will be 11MHz.
BCLKIN	in	**Bus Clock input**. Allows the EBC to monitor the BCLK signal.

I/O Recovery

The ISA bus's default ready timer built into the EBC automatically forces accesses to 8 or 16-bit ISA I/O devices to append one wait state to the bus cycle. If a delay of longer than one wait state is desired, the signal LIOWAIT#, Long I/O Wait, may be asserted to provide a maximum of eleven wait states when accessing 8-bit ISA I/O slaves or three wait states when accessing 16-bit ISA I/O slaves.

Testing

Normally pulled high with an external pullup resistor, an active level on the TEST1# input causes the EBC to float all of its outputs except BCLK. This allows a board tester to gain control of all of the output signal lines for testing purposes.

ISP interface unit

The ISP interface unit provides the interconnect between the EBC and the ISP chip. For a description of the signals involved, refer to the next section.

82357 Integrated System Peripheral (ISP)

Introduction

The majority of the logic contained within the Integrated System Peripheral, or ISP, was described in detail earlier in this publication and in the MindShare book entitled *ISA System Architecture*. The information presented here is intended as a summary of the functions present in the ISP. Where applicable, signals indigenous to the ISP are described. Figure 12-6 provides a detailed view of the major logic blocks contained within the ISP.

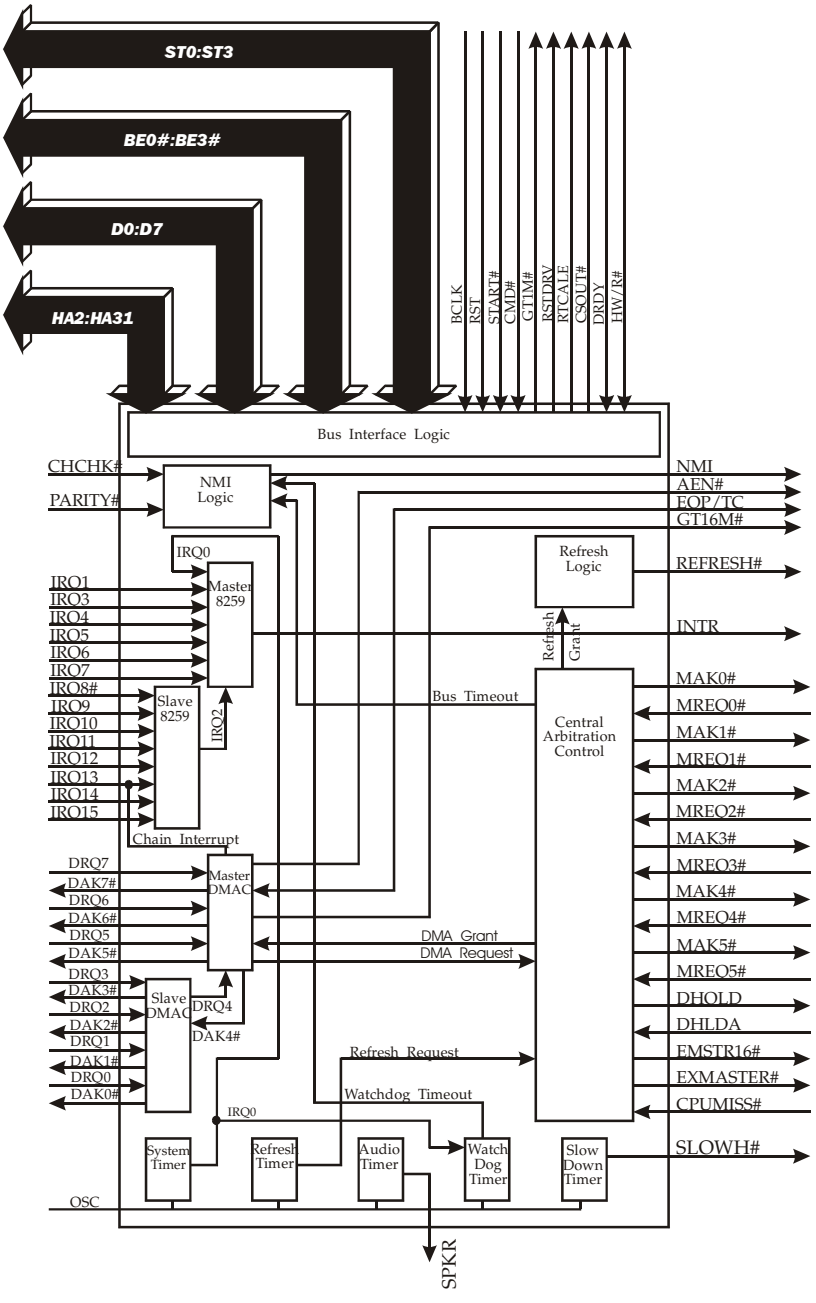


Figure 12-6. The ISP Block Diagram

NMI Logic

In an EISA system, there are four possible hardware causes for generation of NMI to the host CPU:

- A Channel Check, or CHCHK#, from an ISA or EISA card reporting a catastrophic failure.
- A system board RAM parity error (PARITY#).
- A Watchdog Timer timeout because an applications program has disable interrupt recognition for an extended period of time.
- A bus Timeout from the Central Arbitration Control because the current bus master has refused to yield the buses within the allowed period of time (8 microseconds for an EISA bus master or 2.5 microseconds for a DMA channel).

The programmer may also force the NMI logic to generate an NMI by writing to I/O port 0462h with any data.

Interrupt Controllers

The ISP contains two modified Intel 8259A Programmable Interrupt Controllers in a master/slave configuration. Together, they provide a total of fifteen interrupt request lines. Eleven of these are attached to the EISA/ISA card slots, while the remainder are reserved for special system board functions. The Interrupt Acknowledge input to the ISP is conspicuous by its absence. When a bus master other than the DMAC or the Refresh logic is bus master, the ST2 signal line is an input to the ISP and it performs the interrupt acknowledge function. Whenever the EBC detects an interrupt acknowledge bus cycle on the host bus, it sets ST2 low to signal interrupt acknowledge to the interrupt controllers in the ISP.

Two new registers have been added to allow individual programming of each interrupt request input as level-sensitive or edge-triggered. They are referred to as the ELCR, or Edge/Level Control registers. The master interrupt controller's ELCR resides at I/O port 04D0h, while the slave's resides at I/O port 04D1h. Bit zero in the master's ELCR corresponds to the IRQ0 input, while bit seven corresponds to the IRQ7 input. Bit zero in the slave's ELCR corresponds to the IRQ8 input, while bit seven corresponds to the IRQ15 input. A zero in a bit position sets up the respective IRQ input to recognize positive, edge-triggered interrupt requests (non-shareable). A one in a bit position sets the IRQ input up

to recognize active low, level-sensitive interrupt requests (shareable). In both ELCR registers, bit 0 must be a zero.

DMA Controllers

The ISP contains two enhanced Intel 8237 DMA Controllers in a master slave configuration. Together, they provide a total of seven DMA channels. DMA channels five through seven may be used by 16-bit I/O devices, while channels zero through three are reserved for 8-bit I/O devices. Each of the DMA channels can be programmed to utilize the following EISA-specific features:

- 8, 16 or 32-bit transfers.
- ISA compatible, Type “A,” Type “B” or Type “C” bus cycles..
- buffer chaining.
- ring buffer.

Detailed information on programming the DMA controllers can be found in the Intel 82350DT EISA Chipset manual.

When a DMA channel becomes bus master, the ISP depends on the EBC to run the bus cycle for the DMA channel. The EBC generates START#, CMD#, IORC# and IOWC#. When a DMA channel becomes bus master, the type of bus cycle to run is indicated by the ISP's ST[3:0] outputs. Table 12-11 defines the ST[3:0] output settings for the different types of DMA bus cycle types.

Table 12-11. Type of DMA Bus Cycle In Progress

ST3	ST2	ST1	ST0	DMA Bus Cycle Type
0	0	0	0	8-bit ISA compatible
0	0	0	1	8-bit Type "A"
0	0	1	0	8-bit Type "B"
0	0	1	1	8-bit Type "C"
0	1	0	0	16-bit ISA compatible
0	1	0	1	16-bit Type "A"
0	1	1	0	16-bit Type "B"
0	1	1	1	16-bit Type "C"
1	0	0	0	32-bit ISA compatible
1	0	0	1	32-bit Type "A"
1	0	1	0	32-bit Type "B"
1	0	1	1	32-bit Type "C"
1	1	x	x	DMA controller Idle

System Timers

The ISP contains five programmable system timers necessary to the proper operation of any EISA machine. All of these timers derive their timing from the ISP's OSC input signal of 1.19318MHz.

- The System Timer is programmed during the POST to output a pulse onto IRQ0 once every 55ms.
- The Refresh Timer is programmed during the POST to output a Refresh Request to the Central Arbitration Control once every 15.09 microseconds.
- The Audio Timer is programmed by an applications program to yield the desired output frequency on the SPKR output to the speaker driver on the system board.
- The Watchdog Timer may be utilized by multitasking operating systems to detect a cessation of interrupt servicing. The Watchdog Timer counts unserviced IRQ0 output pulses from the System Timer. When its initial count is exhausted, the Watchdog Timer generates a Watchdog Timeout to the NMI logic, causing it to generate NMI to the host CPU.
- The Slowdown Timer allows the programmer to make the host CPU appear to run slower to facilitate the proper operation of game software and some copy protection schemes. The Slowdown Timer and all of the other timers are described in the MindShare book entitled *ISA System Architecture*.

Central Arbitration Control

The ISP incorporates the Central Arbitration Control, or CAC. The operation of the CAC is described earlier in this publication. In the event of a Bus Timeout (when a bus master refuses to yield control of the buses within a time limit), an NMI is generated to the host CPU. In order to force the errant bus master off the bus, the CAC sets the ISP's RSTDRV output active to reset the bus master. The buses are then granted to the host CPU so it can service the NMI. In the NMI interrupt service routine, the programmer may read the contents of the Bus Master Status Latch at I/O port 0464h to determine the identity of the faulty bus master card. Bits zero through five in this register indicate which EISA bus master card was last granted the buses. Bit zero corresponds to the bus master in EISA card slot one, while bit five corresponds to the bus master in EISA card slot six. Bits six and seven in this register aren't used.

Refresh Logic

The Refresh Logic is contained in the ISP. It arbitrates for the buses once every 15.09 microseconds when the Refresh Timer sets the internal signal Refresh Request active. The CAC uses an internal Refresh Grant line to grant the buses to the Refresh Logic. At that time, the Refresh logic sets the ISP's REFRESH# output active. The Refresh logic drives the row address onto the host address bus, HA[31:2].

Miscellaneous Interface Signals

Table 12-12 defines ISP signals not defined elsewhere.

Table 12-12. Miscellaneous ISP Signals

Signal	Direction	Description
CPUMISS#	in	Generated by the host CPU logic, it indicates that the host CPU or its related cache controller requires the use of the buses to run a bus cycle. This is an input to the Central Arbitration Control.
EXMASTER#	out	**EISA Master**. Generated by the Central Arbitration Control when the buses are granted to an EISA bus master. This output is connected to the EBC so it will know whether an EISA bus master is performing a bus cycle.
EMSTR16#	out	**Early 16-bit Bus Master**. Set active by the Central Arbitration Control if the buses are granted to a 16-bit ISA bus master. This output is connected to the EBC so it will know whether an ISA bus master is performing a bus cycle.
DHOLD	out	**Hold Request**. When the Central Arbitration Control is going to grant the buses to a device other than the host CPU, it must first force the host CPU to relinquish control of the buses. The ISP sets its DHOLD output active to the EBC. The EBC, in turn, sets HHOLD (Host Hold Request) active. HHOLD is connected to either the host CPU's (in a cacheless system) or the host cache controller's HOLD line. This forces the host off the bus. In response, the host sets HHLDA, Host Hold Acknowledge, active to the EBC. The EBC, in turn, sets its DHLDA output active to the ISP to inform the CAC that the host is off the bus.
DHLDA	in	**Hold Acknowledge**. See DHOLD description.
GT16M#	out	**Greater Than 16MB**. Generated by the DMA logic in the ISP if the active DMA channel is driving a memory address greater than 16MB (address greater than 00FFFFFFh) onto the host address bus, HA[31:2]. GT16M# is sent to the EBC's ISA interface unit, where it determines whether the MRDC# or the MWTC# will be set active during the DMA bus cycle. See description of HGT16M# in table 12-6.

Table 12 - 12, cont.

Signal	Direction	Description
EOP/TC	in/out	**End-of-Process or Transfer Complete**. Generated by the DMA controller at the end of a DMA transfer when the transfer count has been exhausted. The TC signal is connected to all EISA/ISA slots. When TC is detected by the I/O device associated with the DMA channel, the I/O device will respond by setting its respective IRQ line active to signal the end of the transfer. EISA I/O cards may also generate TC to the DMA controller to prematurely terminate a transfer (e.g., in the case of an error). TC also is used to inform Bus Masters when to reprogram the DMA address buffer when buffer chaining is used. This only pertains to bus masters that program the DMA channel for buffer chaining.
AEN#	out	**Address Enable**. Generated by the DMA controller whenever a DMA channel is bus master and is driving a memory address onto the host address bus, HA[31:2]. For more information, refer to the chapter entitled "EISA Automatic Configuration."
DRDY	in/out	When an ISA bus master is accessing one of the registers within the ISP, this acts as the ready line and is connected externally to the CHRDY signal. When a DMA channel is bus master, DRDY acts as the ready input from the I/O slave associated with the active DMA channel. This permits the I/O slave to lengthen a bus cycle until it is ready to complete it.
CSOUT#	out	**Chip-Select Out**. Whenever any bus master accesses any of the ISP's internal registers, the signal CSOUT# is set active. It should act as an enable for an external data bus buffer between the ISP and data EBB. The direction of the buffer is dictated by whether a read or a write transaction is in progress.
RTCALE	out	**Real-Time Clock Address Latch Enable**. Any write to the Real-Time Clock chip's address port at I/O address 0070h will cause RTCALE to be set active. This signal informs the RTC chip that a CMOS RAM address is present on the data bus and should be latched.

Table 12 - 12, cont.

GT1M#	out	**Greater Than 1MB**. The ISP contains a memory address decoder designed to recognize any memory address less than 1MB (in the 00000000h through 000FFFFFh range). GT1M# is set active whenever the memory address is greater than 000FFFFFh. The state of this signal is used within the EBC's ISA interface unit to determine whether or not to set the SMRDC# or SMWTC# signal active. If the address is below 1MB, SMRDC# or SMWTC# should be set active.

32-bit EISA bus master EISA-based systems support 32-bit EISA bus master cards. A bus master card typically includes an on-board processor and local memory. It can relieve the burden on the main processor by performing sophisticated memory access functions, such as scatter/gather block data transfers.

82350DT EISA chip set The Intel 82350DT EISA chip set. The primary chips used by most manufacturers includes the 82358DT EISA Bus Controller, or EBC, the 82357 Integrated Systems Peripheral, or ISP, and the 82352 EISA Bus Buffers, or EBBs

82352 EISA Bus Buffer Part of the Intel 82350 EISA chip set used for two separate functions: one for the address latching and buffering and one for the data buffering and steering.

82357 ISP This chip is part of the Intel 82350 chip set and contains a variety of functions including: the DMA controllers, Interrupt controllers, Timers, Arbitration logic, and NMI logic.

8237 DMACs The Intel DMA controllers used in ISA systems.

AEN The signal used in ISA systems to disable all I/O address decoders so they do not respond to a DMA address. Also used in EISA systems to independently enable I/O address decoders

AEN logic Logic responsible for controlling the AEN signal so that DMA cycles, standard access to ISA expansion devices and slot specific I/O addressing occur properly.

Address translation The process of converting one type of address to another. For example: translating the address from an ISA Bus Master (SA0:SA16, LA17:LA23 and BHE#) to a 32-bit address (LA2:LA31 and BE0:BE3) required by 32-bit EISA devices.

Arbitration Efficient bus sharing among the main CPU, multiple EISA bus master cards and DMA channels according to a priority scheme.

Arbitration scheme EISA uses a three-way rotational priority scheme between the Refresh Logic, CPU and Bus Masters (shared), and DMA Channels.

BALE An ISA bus signal that is a buffered version of ALE. This signal is used by expansion devices to notify them that a valid address is on the ISA bus.

BCLK An ISA bus signal (bus clock) that provides the timing reference for all bus transactions.

BCPR Services The legal firm that manages the EISA specification.

Bridge The EISA chip set must allow the addresses and data generated by a bus master to propagate onto all of the system buses so all of the devices in the system can be communicated with. The connection between buses is termed a bridge.

Buffer chaining A DMA function that permits the implementation of scatter write and gather read operations. A scatter write operation is one in which a contiguous block of data is read from an I/O device and is written to two or more areas of memory, or buffers. A gather read operation reads a stream of data from several blocks of memory, or buffers, and writes it to an I/O device.

Burst bus cycle A burst transfer is used to transfer blocks of data between the current bus master (or DMA device) and EISA memory. After the initial transfer in a block data transfer, each subsequent EISA Burst bus transfer can be completed in one BCLK period

Burst DMA A DMA bus cycle that supports burst.

Bus Arbitration A process that determines how bus sharing among the main CPU, multiple EISA bus master cards and DMA channels is handled.

Bus Arbitration Scheme. See arbitration scheme

Bus Arbitration Signals. The signals available on the EISA bus that are used by bus masters to gain ownership of the buses. A pair of signals, MASTER REQUEST and a MASTER ACKNOWLEDGE exist for each bus master.

Bus Cycle Definition Specifies the type of bus cycle being run. Memory read, memory write, I/O read, I/O write, Interrupt acknowledge, Halt or Shutdown.

Bus cycle, EISA std. Standard EISA bus cycle. A bus cycle based on a default a zero wait-state operation over the EISA bus.

Bus master priority The priority a bus master has in the rotational scheme. The priority changes as bus masters gain control of the buses.

Bus timeout Upon being preempted by removal of its Acknowledge, the current bus master must relinquish control of the buses within a prescribed period of time. Failure to do so results in a bus timeout.

Cache controller A cache memory controller maintains copies of frequently accessed information read from DRAM memory in the cache.

Central Arbitration Control. The logic responsible for managing the bus arbitration process.

Command translation The process of translating between EISA and ISA type commands.

CMD# CMD# is an EISA signal that is set active by the system board coincidentally with the trailing edge of START#. Only the system board drives the CMD# line. CMD# then remains active until the end of the bus cycle.

Configuration file A file for each expansion card that describes the programmable options available on the card. Used in the EISA automatic configuration process.

Configuration Process A process that uses information provided by EISA expansion board manufactures and the system manufacturer to configure the system for conflict free operation.

Data bus The group of signal lines used to transfer data between devices.

Data Bus Steering A process used to ensure data travels over the correct paths between the current bus master and the currently addressed device.

DMA burst bus cycles DMA bus cycles that supports burst.

DMA cascade channel The DMA cascade channel connects (cascades) two DMA Controllers together. DMA channel 4 is used as the cascade channel.

DMA clock The clock used by the DMA Controller to control its data transfer timing. DMA clock also called DCLK is typically one-half the speed of BCLK.

DMA controller The devices used to perform the DMA transfers in an EISA system. Two modified 8237 DMA controllers are cascaded together to provide support for seven EISA DMA channels.

DMA devices An I/O device that supports DMA transfers.

DMA Extended Write A option associated with DMA bus cycle timing that extends the amount of time that the read command line is active.

DMA Page Register Each DMA channel has an external Page Register used to provide additional address capability. The DMA Controller natively only has the ability to handle 64KB of memory locations.

DMA, Type A bus cycle. DMA bus cycle type that transfers data at a rate of every six BCLK periods.

DMA, Type B bus cycle. DMA bus cycle type that transfers data at a rate of every four BCLK periods.

DMA, Type C bus cycle. See burst bus cycle.

Downshift burst A burst bus cycle performed by a 32-bit EISA bus master when communicating to a 16-bit EISA slave that supports burst.

EBB See EISA bus buffer

EBC See EISA bus controller

EISA bus buffer Two EISA bus buffers (EBBs) are typically used in EISA systems: the Data EBB and the Address EBB.

The Data EBB controls the data transceivers when routing data between the host and EISA buses and performs data bus steering when necessary, utilizing latches and data bus transceivers.

The Address EBB ensures that the address generated by the current bus master is seen by every host, EISA and ISA slave in the system.

EISA bus controller Together with the Data and Address EBBs, the EBC provides the bridging, translation and data bus steering functions

Edge/level control register Allows each interrupt request input to the interrupt controller to be programmed to recognize either edge trigger for ISA devices or level triggering for sharable EISA devices.

ELCR See Edge/Level Control Register.

EX16# EISA size 16 signal that specifies that a 16-bit EISA device is being addressed.

EX32# EISA size 32 signal that specifies that a 32-bit EISA device is being addressed.

EXRDY Used by EISA devices to stretch the default timing beyond zero wait-states if the device's access time exceeds the default ready timing.

HLDA See Hold Acknowledge

HOLD See Hold Request

Hold Acknowledge Hold acknowledge. A microprocessor output that notifies the request device that the microprocessor has given up ownership of the buses.

Hold Request Hold request. A microprocessor input that is used by bus masters to gain ownership of the buses.

Host bus	The bus on which the main CPU and main memory reside.
Peripheral	A chip in the EISA chip set (ISP) that contains a variety of functions including; the interrupt controllers, DMA controllers, arbitration logic, timers, and NMI logic.
Interrupt acknowledge	A signal sent to the interrupt controller to indicate that its request is being acknowledged.
Interrupt latency	The time that expires between a device requesting service via an interrupt request and when the servicing finally occurs.
Interrupts, phantom	An erroneous interrupt triggered at the input of the interrupt controller, usually caused by a noise spike.
Interrupts, shareable	The ability of two devices to share a single interrupt request line (IRQ) and operate without conflict.
LA bus	Latchable Address bus. A portion of the ISA bus that connects to 16-bit devices. These address lines are valid earlier that the System Address lines (SA) and provide the ability of 16-bit devices to operate at zero ISA wait-states.
LOCK# signal	Bus Lock. Prevents other bus masters from gaining control of the EISA bus when the current master asserts LOCK# when performing read/modify write operations.
M/IO#	Memory or I/O signal. Used by EISA devices to either specify or determine whether the address currently on the EISA bus is for a memory or I/O device. Also an output from 386 and 486 microprocessors.
MSBURST#	Master Burst signal. Asserted by EISA masters to inform a bursting slave that a burst cycle will be run.
NMI	Non-maskable Interrupt. Used to report serious error conditions to the microprocessor.
Preemption	The ability of bus masters to request and gain ownership of the system buses from the current bus master.

Refresh The process of keeping dynamic memory from loosing information from the bit cell due to capacitor discharge. All DRAM throughout the system is refreshed approximately every fifteen microseconds.

Refresh logic The logic that runs refresh bus cycles. The refresh logic is a bus master capable of gaining ownership of the buses on a regular basis.

Ring buffers A ring buffer reserves a fixed range of memory to be used for a DMA channel. Once the buffer has been filled, data can be stored at the beginning of the buffer again and old information can be over-written if it has already been read by the microprocessor.

Rotating priority A three-way rotational priority scheme between the Refresh Logic, CPU and Bus Masters (shared), and DMA Channels to determine which bus master will be next granted use of the buses.

Slave A term used to refer to target devices with which bus masters communicate in an EISA system.

SLBURST# Slave burst signal. Used by EISA bursting slaves when addressed to notify the current bus master that they support burst cycles.

Slot-specific I/O The I/O addressing method used by EISA providing independent address space on a slot-by-slot basis to support automatic expansion board configuration.

START# The EISA signal that goes active at the beginning of address time (T1) and inactive at the end of address time. Asserted by the current bus master.

System timers The timers that are standard with all EISA systems and are contained in the ISP. These timers include the system timer (0), refresh timer, speaker timer, watchdog timer, and slowdown timer.

Type A DMA bus cycle. See DMA, Type A

Type B DMA bus cycle. See DMA, Type B

Type C DMA bus cycle. See DMA, Type C

W/R# Write or read. Used by EISA devices to either specify or determine whether the current EISA bus cycle is a write or read operation. Also an output from 386 and 486 microprocessors.

—E—

—F—

—G—

—K—

—L—

—M—

—T—

—W—

—X—