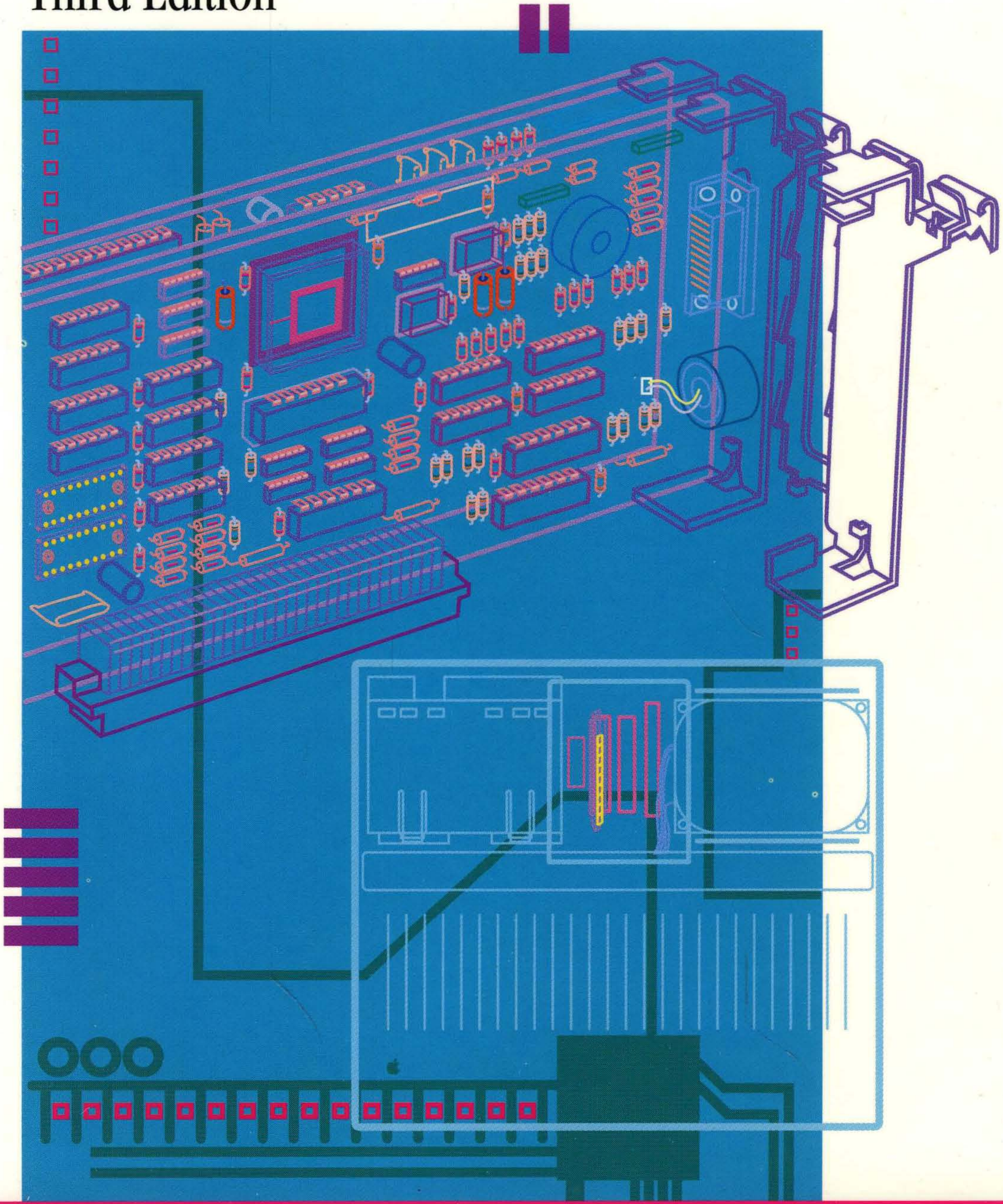
Designing Cards and Drivers for the Macintosh Family

by Apple Computer, Inc.

Third Edition



Designing Cards and Drivers for the Macintosh Family

Third Edition

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

APPLE COMPUTER, INC.

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, CA 95014-6299
408-996-1010

Simultaneously published in the United States and Canada.

ISBN 0-201-60855-3
1 2 3 4 5 6 7 8 9-MU-9695949392
First printing, April 1992

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures and tables

14 Electrical Design Guide for 68020 Direct Slot Expansion Cards / 303

15 Electrical Design Guide for 68030 Direct Slot Expansion Cards / 315

Preface **About This Book**

The purpose of this book is to provide you, the developer, with the information that you need to develop expansion cards and device drivers for the Apple Macintosh family of computers. The introduction to this book discusses the Macintosh-family expansion strategy. It will give you an insight into Apple's plans for current and future hardware expansion for the Macintosh computer family. Following the introduction, the book is divided into three parts.

Part I defines the specifications of the NuBus™ expansion interface, provides electrical and mechanical guidelines for designing NuBus expansion cards, and supplies information that is vital to the design of driver software.

Part II is devoted to the processor-direct slot (PDS) expansion interface. This part defines the design criteria and provides electrical and mechanical guidelines for designing expansion cards for Macintosh computers with processor-direct slots.

Part III gives design specifications and provides electrical and mechanical guidelines for expansion interfaces that have only one specific purpose.

Design philosophy

In keeping with the Macintosh design philosophy, it is incumbent upon you, the card designer and driver writer, to make the installation of the card and its use by applications as transparent as possible. To the greatest extent possible, an application should rely on only a few high-level calls (if any) and not have to use low-level calls. To do otherwise jeopardizes the broadest potential use of your product.

Conventions used in this book

The following visual cues are used throughout this manual to identify different types of information:

◆ *Note:* A note like this contains information that is interesting but is not essential for an understanding of the main text.

△ **Important** A note like this contains information that is essential for an understanding of the main text. △

▲ **Warning** A warning like this indicates a potential problem. ▲

When new or specialized terms are defined, they appear in **boldface.** Those terms are also defined in the glossary at the back of the book. The glossary contains additional terms of interest that are not boldfaced in the text.

Hexadecimal numbers are preceded by a dollar sign ($). For example, the hexadecimal equivalent of decimal number 16 is written as $10.

In Part I, a NuBus word consists of 32 bits and a NuBus halfword consists of 16 bits. In Part II, a word consists of 16 bits and a longword consists of 32 bits. The two parts follow a different convention for their terminology to be consistent with the outside documentation to which each part is related: the Texas Instruments specification of the NuBus for Part I and the Motorola documentation for the MC68000, MC68020, MC68030, and MC68040 microprocessors for Part II.

Address ranges are given as "*lower address* through *higher address*" or "*lower address–higher address*"; in either form the range is inclusive of the given endpoints. For example, an access range in memory is given in text as "$00 0000 through $3F FFFF," and in a table as "$00 0000–$3F FFFF."

A preceding slash is used to designate an active-low signal, for example, /ACK. A range of signals is designated like this, with the highest-numbered signal first: /AD31–/AD0. If there is more than one subrange in a set, the subranges are enclosed in angle brackets like this: </AD31–/AD29, /AD7–/AD0>.

Macintosh resource types are designated by enclosing them in single straight quotation marks, for example, `'INIT'`.

The term *processor* is often used instead of *microprocessor* or *CPU. Processor* usually refers to the primary microprocessor on the main logic board, and *coprocessor* refers to an auxiliary processor such as the MC68882 floating-point unit on the main logic board or another processor on an expansion card.

The terms *processor-direct slot, PDS, 68000 Direct Slot, 68020 Direct Slot, 68030 Direct Slot,* and *68040 Direct Slot* are all used to identify the processor-direct expansion interface associated with some Macintosh computers. Other documents may use a term such as *processor dependent slot* to identify this interface.

The following abbreviations are used:

K	1024
GB	gigabyte
Kbit	kilobit
KB	kilobyte
kΩ	kilohm
Mbit	megabit
MB	megabyte
µA	microampere
µs	microsecond
mA	milliampere
ms	millisecond
ns	nanosecond
Ω	ohm
pF	picofarad
RMS	root mean square

The distinction between boards and cards is as follows: boards are a permanent part of the computer (for example, the main logic board), whereas cards are insertable and can be added or exchanged for functional expansion or reconfiguration of the system.

The Courier font

Throughout the book, the names of specific software structures or fields within a structure are in the `Courier` font.

For example, suppose you see this sentence:

In the example of the `SExecBlock` data type, the `RevisionLevel` field is always 02, the `reserved` field is always 00, and the `CPUID` field identifies the processor—01 for the 68000, 02 for the 68020, 03 for the 68030, and 04 for the 68040.

The word `SExecBlock` is in the `Courier` font to indicate that it is the name of a structure. The words `RevisionLevel`, `reserved`, and `CPUID` are in the `Courier` font to indicate that they are fields within the `SExecBlock` structure.

About the mechanical drawings and design guides

Mechanical drawings of cards and connectors are provided in several chapters and in foldouts in the back of the book. Some of these drawings are design guides used within Apple Computer and were correct at the time of publication; they are, however, subject to change in the future.

About the Macintosh technical documentation

Apple Computer, Inc., provides a suite of technical books that explain the hardware and software of the Macintosh family of computers.

The original Macintosh documentation consisted of the first three volumes of *Inside Macintosh.* Shortly after the introduction of the Macintosh Plus (with the 128 KB ROM), Volume IV of *Inside Macintosh* was released as a delta guide. That is, Volume IV covered only those aspects of the Macintosh Plus that were different from those in earlier Macintosh computers. Later, a fifth volume was added, called *Inside Macintosh,* Volume V. It is also a delta guide, covering the new and different features of the Macintosh SE and the Macintosh II computers. The latest volume, *Inside Macintosh,* Volume VI, describes the System 7 environment.

As the variety and the sophistication of Macintosh computers evolve, so does the documentation. To provide information that is comprehensive—and that provides answers to specific questions—Apple provides a whole family of books. Each of these books gives complete information about a single subject and may include some information that also appears in *Inside Macintosh. Guide to the Macintosh Family Hardware,* second edition, and this book are two of the books in this family.

For programmers and developers who are new to the Macintosh world, Apple has created two introductory books: *Technical Introduction to the Macintosh Family* and *Programmer's Introduction to the Macintosh Family.*

In addition to the books about the Macintosh itself, there are books on related subjects. Examples are a book about the user interface, a book about Apple's floating-point numerics, and the reference books for the Macintosh Programmer's Workshop.

Table P-1 gives a brief description of many of the books in the Macintosh technical documentation.

■ **Table P-1** Macintosh technical documentation

Technical documentation	Reference material
Inside Macintosh	
Inside Macintosh, Volumes I–III	Complete reference to the Macintosh Toolbox and Operating System for the original 64 KB ROM
Inside Macintosh, Volume IV	Delta guide to the Macintosh Plus (128 KB ROM)
Inside Macintosh, Volume V	Delta guide to the Macintosh SE and Macintosh II (256 KB ROM)
Inside Macintosh, Volume VI	Description of System 7
Inside Macintosh X-Ref, *revised edition*	A single general index to eleven technical reference books for the Apple Macintosh family of computers: *Inside Macintosh,* Volumes I–VI; *Inside the Macintosh Communications Toolbox; Programmers Introduction to the Macintosh Family; Technical Introduction to the Macintosh Family; Designing Cards and Drivers for the Macintosh Family,* second edition; and *Guide to the Macintosh Family Hardware,* second edition
Introductory books	
Technical Introduction to the Macintosh Family	Introduction to the Macintosh software and hardware; explains concepts and terminology that are specific to the Macintosh family of computers.
Programmer's Introduction to the Macintosh Family	Introduction to programming the Macintosh system for programmers who are new to it
Single-subject books	
Designing Cards and Drivers for the Macintosh Family	Hardware and device-driver reference for the expansion capabilities of the Macintosh computer family
Guide to the Macintosh Family Hardware	Hardware reference and developer's guide for the Macintosh computer family

(continued)

■ **Table P-1** Macintosh technical documentation (continued)

Technical documentation	Reference material
Related books	
Human Interface Guidelines: The Apple Desktop Interface	Detailed guidelines for developers implementing the Macintosh user interface
Apple Numerics Manual	Description of the Standard Apple Numerics Environment (SANE), an IEEE-standard floating-point environment supported by all Apple computers
Macintosh Programmer's Workshop 3.0 Reference	Workshop (MPW), Apple's software computers development environment for all Macintosh
Apple Developer Notes	
Macintosh IIsi, LC,and Classic DeveloperNotes, APDA publication number *M0991LL/A*	Interim hardware reference and developer's guide for the Macintosh IIsi, Macintosh LC, and Macintosh Classic computers; obsolete when the information has been incorporated into the third edition of *Guide to the Macintosh Family Hardware*
Macintosh Classic II, MacintoshPowerBook Family, and Macintosh Quadra Family Developer Notes, APDA publication number *R0143LL/A*	Interim hardware reference and developer's guide for the Macintosh Classic II, Macintosh PowerBook, and Macintosh Quadra computers; obsolete when the information has been incorporated into the third edition of *Guide to the Macintosh Family Hardware*

How to get more information

Several organizations exist that provide support for Macintosh hardware and software developers. This section tells you how to contact APDA, Apple user groups, and Apple Developer Services.

APDA

APDA offers convenient worldwide access to over three hundred development tools, resources, and training products, and to information for anyone interested in developing applications on Apple platforms. Customers receive the quarterly *APDA Tools Catalog,* featuring the most current version of Apple development tools and the most popular third-party development tools. Ordering is easy; there are no membership fees, and application forms are not required for most of our products. APDA offers convenient payment and shipping options, including site licensing.

To order products or get additional information, contact

APDA
Apple Computer, Inc.
20525 Mariani Avenue, M/S 33-G
Cupertino, CA 95014-6299

800-282-2732 (United States)
800-637-0029 (Canada)
408-562-3910 (International)
Fax: 1-408-562-3971
Telex: 171-576
AppleLink address: APDA

User groups

Apple user groups are associations of individuals who share information about Apple computers and related products. For information about Apple user groups in your area, call this toll-free number:

800-538-9696

Ask for extension 500.

Apple Developer Services

Apple's goal is to provide developers with the resources they need to create new Apple-compatible products. Apple offers two programs: the Partners Program, for developers who intend to resell Apple-compatible products; and the Associates Program, for developers who do not intend to resell Apple-compatible products and for other people involved in the development of Apple-compatible products.

As an Apple Partner or Associate, you will receive monthly mailings including a newsletter, Apple II and Macintosh Technical Notes, pertinent Developer Program information, and all the latest news relating to Apple products. You will also receive the *Macintosh Services Directory* and automatic membership in APDA. You'll have access to developer AppleLink and to Apple's Developer Hotline for general developer information.

As an Apple Partner, you'll be eligible for discounts on equipment and you'll receive technical assistance from the staff of Apple's Developer Technical Support department.

For more information about Apple's developer support programs, contact Apple Developer Programs at the following address:

Apple Developer Programs
Apple Computer, Inc.
20525 Mariani Avenue, M/S 51-W
Cupertino, CA 95014-6299

Introduction Expansion Strategy for the Macintosh Family

Apple has decided on an expansion strategy that limits the Apple Macintosh family of computers to three distinctly different internal architectural expansion configurations: the NuBus™ expansion interface, the processor-direct slot (PDS) expansion interface, and the application-specific expansion interface. Limiting the expansion architecture to three categories ensures that expansion card developers, both internal and external to Apple, have some degree of predictability and stability in their expansion card designs. Since Apple depends upon you, the third-party hardware developer, to create the expansion cards that enhance many Macintosh computers, it is important that you are aware of this expansion strategy. This section gives you the information you will need to make good decisions on what cards to develop, and for what Macintosh models, both present and future.

Limiting the number of expansion interfaces

Apple's implementation of NuBus represents a mature expansion mechanism that has been adopted as the primary expansion vehicle for the Macintosh family of modular computers and can be supported across a variety of Macintosh products.

Macintosh computers such as the Macintosh SE, the Macintosh SE/30, and the Macintosh LC use the processor-direct slot (PDS) expansion interface. The Macintosh IIfx, the Macintosh Quadra 700, and the Macintosh Quadra 900 all have a processor-direct slot, but their primary expansion interface is the NuBus. The Macintosh IIsi has only one expansion slot, but it is unique in that it supports either a NuBus card or a PDS card, depending on which adapter card is installed. The Macintosh Portable also has a processor-direct slot, but its usefulness is somewhat limited in comparison to that of the compact computers. You can think of the PDS as an extension of the microprocessor used in a particular Macintosh model. Because of this dependency on the microprocessor, the slot configuration changes whenever a newer, more powerful processor is adopted. For example, the MC68000 microprocessor in the Macintosh SE uses a 96-pin PDS connector, while the PDS connector in the Macintosh SE/30 has been expanded to 120 pins to take advantage of the enhanced features of the MC68030 microprocessor. Therefore, in addition to NuBus, the minimum number of processor-direct slot expansion interfaces that you have to support is determined by the number of different microprocessors that are implemented in the Macintosh family of computers.

Apple plans to limit the number of expansion interfaces by adopting a slot specification for each microprocessor that is sufficiently comprehensive to apply to most of the Macintosh computers that use the same microprocessor. You may find, however, that because of electrical and physical design constraints, a card designed for one Macintosh will not work in another Macintosh even though both computers use the same microprocessor.

The application-specific category refers to expansion interfaces that are dedicated to a singular, unique purpose. Usually computers that provide this feature also have NuBus or a processor-direct slot as their primary means of expansion. For example, in addition to its processor-direct slot, the Macintosh Portable includes three other expansion connectors: one for a ROM card, one for a RAM card, and one for a modem card. Although NuBus is the primary means of expansion for the Macintosh IIci, this machine includes an expansion interface connector designed specifically for a cache memory card. In some specific applications, you might find that the expansion connector is physically identical to the connector used for the processor-direct slot, but it will probably not provide the same functions.

NuBus expansion

The NuBus is Apple's primary expansion interface for Macintosh computer products. It is available in configurations of up to six identical slot connectors. The NuBus is a truly powerful expansion vehicle providing features such as a small pin count, a large area for card implementation, a versatile bus protocol, high data-transfer rates, variably sized data transfers, and parallel bus arbitration.

The NuBus hardware requires a large space within the Macintosh case and usually requires some additional circuitry. Therefore NuBus is inappropriate for compact designs such as the Macintosh SE and the Macintosh SE/30. These designs are better suited for PDS expansion.

Newer, more powerful Macintosh computers, such as the Macintosh Quadra 700 and the Macintosh Quadra 900, support slave block-transfer modes and other NuBus enhancements that were not supported on earlier Macintosh models. These enhancements will not affect the capabilities of your current card designs, but will add more usability to future designs.

Apple is committed to the NuBus expansion interface being the primary expansion system for the Macintosh family and will continue to support it in the foreseeable future. For a detailed description of the NuBus specification, as well as guidelines for designing NuBus expansion cards, see Part I, "The NuBus Expansion Interface."

Processor-direct slot (PDS) expansion

Apple uses the **processor-direct slot (PDS)** expansion interface on compact, or small-footprint, Macintosh computers such as the Macintosh SE, the Macintosh SE/30, and the Macintosh LC, or any design for which NuBus is inappropriate. The Macintosh IIfx, the Macintosh Quadra 700, and the Macintosh Quadra 900 also include a processor-direct slot, but their primary means of expansion is the NuBus interface. A PDS implementation brings the microprocessor address, control, and data lines, along with clock, power, and a few model-specific signals, to a single expansion connector on the main logic board. The Macintosh Portable has a processor-direct slot, but only limited power is available from the expansion connector.

An advantage of the PDS interface is that it provides an expansion mechanism that does not burden the average user, who may not need the extensive expansion capabilities of the NuBus configuration. Also, you can design a PDS expansion card with a smaller form factor than a NuBus card, and since no additional circuitry is usually required for PDS expansion, it costs less to implement than NuBus. Finally, a PDS expansion card has direct access to the microprocessor, resulting in a speed advantage that allows support of some tasks that cannot be done with a NuBus card.

A disadvantage of the PDS expansion interface is its inability to support the bus structure across Macintosh products that use different microprocessors. Because the PDS expansion interface is an extension of the microprocessor, the configuration of the slot connector will change whenever a newer, more powerful microprocessor is used in the Macintosh family. Other disadvantages include difficulty in extending the bus, the inability to support more than one card, and the requirement that processor activity must be suspended during bus activity.

Apple hopes to limit the number of PDS configurations that you must support. The goal is to have the PDS specification remain constant within a microprocessor family and to have a common physical form factor and electrical characteristics without compromising the Macintosh design. Part II, "The Processor-Direct Slot Expansion Interface," defines the PDS specifications and gives detailed electrical and physical guidelines for designing PDS expansion cards.

The 68000 and 68020 Direct Slot expansion interfaces

The Macintosh SE was the first Macintosh computer offering processor-direct slot expansion. The expansion interface to the MC68000 microprocessor in the Macintosh SE is a 96-pin connector. The 68000 Direct Slot expansion interface is flexible enough to allow you to design coprocessor cards such as accelerators or to extend the I/O capabilities of the computer. The 96-pin expansion connector on the Macintosh Portable is physically identical to that of the Macintosh SE, but the pinout and signals available are different.

The 68020 Direct Slot expansion interface in the Macintosh LC is also a 96-pin connector that is physically identical to that found in the Macintosh SE. However, the pinout and signals are different, so that cards designed for the Macintosh SE or the Macintosh Portable will not work in the Macintosh LC, and vice versa.

The 68030 Direct Slot expansion interface

Both the Macintosh SE/30 and the Macintosh IIfx have a 120-pin expansion connector to satisfy the requirements of the MC68030 microprocessor used in these machines. Once a Direct Slot adapter card is installed in the Macintosh IIsi, it also provides a 120-pin expansion connector. The additional pins allow you to take full advantage of the increased functionality of the processor, including its 32-bit address and data bus capabilities. Although it is possible that the physical form factor could change on future 68030-based machines due to space limitations, the electrical characteristics should remain the same. Due to the difference in physical form factors and electrical characteristics, cards you design for other processors will not work in the 68030 Direct Slot, and vice versa.

The 68040 Direct Slot expansion interface

The Macintosh Quadra 700 and the Macintosh Quadra 900 each have a 140-pin expansion connector to satisfy the requirements of the powerful MC68040 microprocessor used in these machines. The electrical characteristics of the processor-direct expansion connectors of the Macintosh Quadra 700 and the Macintosh Quadra 900 are identical so that cards designed for one 68040-based computer will work in the other one as well. Physically, the size of the processor-direct expansion card for the Macintosh Quadra 700 is the same size as a standard NuBus card, but you can design an oversized expansion card for the Macintosh Quadra 900. Any processor-direct expansion cards that have been designed for 68000-based, 68020-based, or 68030-based Macintosh computers will not work in the Macintosh Quadra 700 and the Macintosh Quadra 900. Likewise, PDS expansion cards designed for the Macintosh Quadra family will not work in other Macintosh models.

Recommended strategy for 68030 and 68040 Direct Slot expansion card design

The 68030 and 68040 Direct Slot electrical specifications contain several types of signals including power, data lines, address lines, control lines, clocks, and machine-specific signals. Most of these signals are classified as *common,* meaning that they will be available on all Macintosh computers that use the 68030 or 68040 Direct Slots. Others, however, are classified as *machine specific,* meaning that they may or may not be present on different Macintosh computers that use the 68030 or 68040 Direct Slots. The intent is, with each new version of the Macintosh, to identify those signals that are common to all machines, to flag those signals that are machine specific, and to provide you with guidelines to know when to use the machine-specific signals. Detailed signal descriptions are provided in Chapter 15, "Electrical Design Guide for 68030 Direct Slot Expansion Cards," and Chapter 16, "Electrical Design Guide for 68040 Direct Slot Expansion Cards."

Because of the scarcity of open areas in the memory maps of new Macintosh PDS computers, you should design your expansion card to occupy an address location corresponding to the 32-bit physical address ranges used by NuBus expansion cards resident in Macintosh computers with the NuBus interface. This method of emulating NuBus address space is called *pseudoslot design.* If you follow the pseudoslot method and design your PDS expansion card along the lines of a NuBus card, the existing Slot Manager ROM firmware controls your card as if it were a NuBus card, the only difference being that the electrical signals arrive through the 68030 or 68040 PDS expansion interface, not the NuBus expansion interface. This means that you can use the same device driver for both your PDS expansion card and its NuBus equivalent. Chapter 15 provides more detailed information on pseudoslot expansion card design.

If you don't take advantage of pseudoslot design, you have to do several things differently. Apple has reserved a range of physical address spaces in the memory maps for 68030 and 68040 Direct Slot cards that do not emulate the NuBus address space. To gain access to the reserved address space, the Macintosh must be in 32-bit mode, and your card driver must be able to switch between 24-bit and 32-bit modes. This means that your card driver must also include specific information to allow access to the card's address space and that the Slot Manager routines cannot be used.

Converting your designs

PDS expansion cards are electrically and mechanically different from NuBus expansion cards. NuBus expansion cards will not work in PDS expansion card slots, and PDS expansion cards will not work in NuBus slots. However, by using the pseudoslot design features in your PDS expansion card, you can easily convert your 68030 or 68040 Direct Slot expansion card to a NuBus card, or vice versa. The fundamental design, with the exception of the bus interface circuitry, can be converted. This is one of the major advantages of using pseudoslot design.

Application-specific expansion

The Macintosh computers that offer application-specific expansion usually have a NuBus or a processor-direct slot as their primary means of expanding the system. By providing an interface that is dedicated to a specific function, you free up a NuBus slot or the processor-direct slot to accept cards that perform a variety of functions such as coprocessing, networking, and so on. The application-specific expansion interface may or may not directly access the processor. The cache memory expansion connector on the Macintosh IIci does, but the ROM, RAM, and modem expansion connectors on the Macintosh Portable, the Macintosh Classic, the Macintosh Classic II, the Macintosh PowerBook 100, the Macintosh PowerBook 140, and the Macintosh PowerBook 170 do not. Part III, "Application-Specific Expansion Interfaces," gives detailed electrical and physical guidelines for designing application-specific expansion cards.

Slot strategy summary

In summary, the preferred expansion mechanism for the Macintosh family is NuBus. The processor-direct slot is used on Macintosh computers with and without NuBus to provide general system expansion. The application-specific expansion interface provides a mechanism for specific functions such as memory expansion.

By designing NuBus cards, you will have access to the rapidly growing installed base of Macintosh computers with NuBus expansion slots. By porting your NuBus design to the 68030 or 68040 Direct Slot via the pseudoslot method, you need to supply only one driver for both the 68030 and 68040 Direct Slot cards and NuBus cards, and you can design cards that will be usable in future Macintosh computers without NuBus.

Part I **The NuBus Expansion Interface**

About Part I

The Apple implementation of NuBus is the subject of Part I of this book. NuBus was originally a Texas Instruments product described in their NuBus specification, document number TI-2242825-0001*A, copyright 1983. The NuBus specification has since become an ANSI (American National Standards Institute) standard. ANSI has published the latest NuBus standard (referred to in this publication as NuBus '90) and distributed it for comment. At the time of this publication, NuBus '90 is a draft standard, and some features in it may change.

Several features of the original NuBus standard, most notably block data transfer and system parity valid, are not supported in the full line of Macintosh computers. Other features of the NuBus '90 draft standard, such as cache coherency, are not currently implemented in any of the Macintosh computers with NuBus. These features have been documented in this section, however, to provide a complete discussion of NuBus. The instances where NuBus features are explained but not implemented are labeled.

Part I contains 11 chapters specifically dedicated to the NuBus expansion interface. Following are brief descriptions of the major topics covered in each chapter.

Chapter 1 provides block diagrams of each Macintosh computer that offers a NuBus expansion interface, compares major features, and gives an overview of computer operation. The chapter then describes the NuBus interface architecture and the state machines used to implement it.

Chapter 2 describes NuBus features, provides a simplified diagram of the NuBus hardware, defines many NuBus terms, classifies the signals used to implement communication over the bus, and discusses the most basic timing and transaction cycle relationships.

Chapter 3 details each signal, its timing, and its line characteristics. The chapter defines various types of bus cycles, then describes the sequential combination of bus cycles to perform transactions.

Chapter 4 gives the rules for arbitration to resolve the contention between cards for bus mastership, so that all cards have access to the bus.

Chapter 5 provides an electrical design guide for NuBus expansion cards, focusing on the electrical requirements of line drivers and receivers.

Chapter 6 provides the physical information you need to design NuBus expansion cards.

Chapter 7 describes how cards in NuBus slots can address memory in a Macintosh computer with the NuBus interface.

Chapter 8 defines the firmware data structures typically stored on the card in ROM.

Chapter 9 describes several driver options, driver installation, and the video driver declaration ROM and routines. Pseudocode for an actual video card driver is provided.

Chapter 10 contains design examples, including schematics and PAL equations for three NuBus cards that have been built and tested.

Chapter 11 concludes Part I of the book with a description of the Macintosh II Video Card.

In the back of this book, following Part III, there are five appendixes. Appendix A provides information on electromagnetic interference (EMI), heat dissipation, and product safety standards and applies to Part I, Part II, and Part III.

Appendix B contains a sample of the declaration ROM firmware code for the Macintosh II Video Card, described in Chapter 8.

Appendix C contains a sample video card driver. This sample code supplements information given in Chapter 9.

Appendix D contains the PAL listings for the NuBus Test Card, described in Chapter 10.

Appendix E contains the PAL listings for the SCSI-NuBus Test Card, described in Chapter 10.

Addressing design philosophy

Whenever possible, use 32-bit addressing conventions and methods. This is your best guarantee of future software compatibility.

NuBus use and licensing requirements

NuBus is a trademark of Texas Instruments, Inc. Part I of this book describes the implementation of NuBus by Apple Computer in Macintosh computers. Certain features of the NuBus interface are not implemented in current Macintosh computers but may be in future products; note is made of that fact where appropriate.

In addition to the NuBus information in this book, you should also refer to the *Standard for a Simple 32-Bit Backplane Bus: NuBus,* ANSI/IEEE Std 1196-1987, and to the NuBus '90 proposal, *Standard for a Simple 32-Bit Backplane Bus: NuBus,* ANSI/IEEE Std 1196-1990. You can order these documents from

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854
908-981-0060

Texas Instruments owns patents on the NuBus. If you wish to make devices for computers with the NuBus interface (including Macintosh computers), you need to obtain a license directly from Texas Instruments. For further details please send your request to

Texas Instruments, Inc.
12501 Research Boulevard
Austin, TX 78759
Attention: NuBus Licensing
M/S 2151

Chapter 1 **Overview of Macintosh Computers With the NuBus Interface**

This chapter provides an overview of the structure and organization of the Macintosh computers that use NuBus as their primary expansion interface. Included in this category are the Macintosh II, the Macintosh IIx, the Macintosh IIcx, the Macintosh IIci, the Macintosh IIfx, the Macintosh IIsi, the Macintosh Quadra 700, and the Macintosh Quadra 900. All use an I/O bus based on the Texas Instruments NuBus to allow expansion beyond the capabilities of the ports (connectors) on the back of the machines; NuBus slots allow a wide variety of devices to be connected. Read this chapter to place the NuBus interface in context within the total computing machine.

Major features

Table 1-1 compares the major features of the Macintosh computers with NuBus.

■ **Table 1-1** Major features of Macintosh computers with the NuBus interface

Feature	Macintosh II	Macintosh IIx	Macintosh IIcx	Macintosh IIci
Processor	MC68020 32-bit address bus 32-bit data bus	MC68030 32-bit address bus 32-bit data bus	MC68030 32-bit address bus 32-bit data bus	MC68030 32-bit address bus 32-bit data bus
Processor clock	15.6672 MHz	15.6672 MHz	15.6672 MHz	25 MHz
Coprocessor	MC68881 floating-point unit	MC68882 floating-point unit	MC68882 floating-point unit	MC68882 floating-point unit
Memory management	24-to-32-bit address translation by Address Management Unit (AMU); or logical-to-physical address translation by optional MC68851 Paged Memory Management Unit (PMMU)	MC68030 has a built-in paged Memory management Unit (PMMU) that allows true 32-bit address translation with hardware page replacement	MC68030 has a built-in PMMU that allows true 32-bit address translation with hardware page replacement	MC68030 has a built-in PMMU that allows true 32-bit address translation with hardware page replacement
Video interface	NuBus video card with on-card screen buffer	NuBus video card with on-card screen buffer	NuBus video card with on-card screen buffer	Built-in video with main memory screen buffer or NuBus video card
RAM	Up to 64 MB of DRAM in eight 30-pin SIMMs	Up to 128 MB of DRAM in eight 30-pin SIMMs	Up to 128 MB of DRAM in eight 30-pin SIMMs	Up to 128 MB of DRAM in eight 30-pin SIMMs; optional DRAM parity with 9-bit SIMMs
ROM	256 KB in four 512 Kbit ROM chips	256 KB standard in 64-pin ROM SIMM, expandable to 64 MB	256 KB standard in 64-pin ROM SIMM, expandable to over 64 MB	512 KB standard in four 1 Mbit ROM chips, optional 64-pin ROM SIMM allows expansion to 64 MB

(continued)

■ **Table 1-1** Major features of Macintosh computers with the NuBus interface (continued)

Feature	Macintosh IIfx	Macintosh IIsi	Macintosh Quadra 700	Macintosh Quadra 900
Processor	MC68030 32-bit address bus 32-bit data bus	MC68030 32-bit address bus 32-bit data bus	MC68040 32-bit address bus 32-bit data bus	MC68040 32-bit address bus 32-bit data bus
Processor clock	40 MHz	20 MHz	25 MHz bus clock 50 MHz CPU clock	25 MHz bus clock 50 MHz CPU clock
Coprocessor	MC68882 floating-point unit	Optional MC68882 available on NuBus and 68030 Direct Slot expansion card adapters	MC68040 has a built-in high-performance floating-point unit	MC68040 has a built-in high-performance floating-point unit
Memory management	MC68030 has a built-in PMMU that allows true 32-bit address translation with hardware page replacement	MC68030 has a built-in PMMU that allows true 32-bit address translation with hardware page replacement	MC68040 has a built-in PMMU that allows true 32-bit address translation with hardware page replacement	MC68040 has a built-in PMMU that allows true 32-bit address translation with hardware page replacement
Video interface	NuBus video card with on-card screen buffer	Built-in video with main memory screen buffer or NuBus video card	Built-in video with VRAM or NuBus video card	Built-in video with VRAM or NuBus video card
RAM	Up to 128 MB of DRAM in 64-pin SIMMs; 32 KB of SRAM, fast RAM cache; optional DRAM parity with 9-bit SIMMs	1 MB soldered to main logic board; up to 64 MB in 64-pin SIMMs; 4 MB DRAM	Up to 20 MB of DRAM in 30-pin SIMMs	Up to 64 MB of DRAM in 30-pin SIMMs, arranged in four separate memory banks
ROM	512 KB standard in one ROM SIMM	512 KB standard in one ROM SIMM	1 MB ROM SIMM, expandable to 4 MB	1 MB ROM SIMM, expandable to 4 MB

(continued)

■ **Table 1-1** Major features of Macintosh computers with the NuBus interface (continued)

Feature	Macintosh II	Macintosh IIx	Macintosh IIcx	Macintosh IIci
Expansion slots	Six NuBus slots	Six NuBus slots	Three NuBus slots	Three NuBus slots, one cache card connector
Keyboard and mouse interface	Two Apple Desktop Bus (ADB) ports	Two ADB ports	Two ADB ports	Two ADB ports
Serial ports	Two mini 8-pin connectors supporting RS-422 and AppleTalk	Two mini 8-pin connectors supporting RS-422 and AppleTalk	Two mini 8-pin connectors supporting RS-422 and AppleTalk	Two mini 8-pin connectors supporting RS-422 and AppleTalk
Floppy disk support	Integrated Woz Machine (IWM) chip controls two internal 800 KB, 3.5" floppy disk drives (one standard, one optional)	Super Woz Integrated Machine (SWIM) chip controls two internal 1.4 MB, 3.5" Apple SuperDrives (one standard, one optional)	SWIM chip controls one internal 1.4 MB, 3.5" SuperDrive and one optional external 1.4 MB SuperDrive	SWIM chip controls one internal 1.4 MB, 3.5" SuperDrive and one optional external 1.4 MB SuperDrive
SCSI ports	One internal 50-pin, one external DB-25	One internal 50-pin, one external DB-25	One internal 50-pin, one external DB-25	One internal 50-pin, one external DB-25
Sound	Apple Sound Chip	Apple Sound Chip	Apple Sound Chip	Apple Sound Chip
Battery	Long-life lithium battery	Long-life lithium battery	Long-life lithium battery	Long-life lithium battery

(continued)

■ **Table 1-1** Major features of Macintosh computers with the NuBus interface (continued)

Feature	Macintosh IIfx	Macintosh IIsi	Macintosh Quadra 700	Macintosh Quadra 900
Expansion slots	Six NuBus slots or one processor-direct slot and five NuBus slots	One NuBus slot or one processor-direct slot	Two NuBus slots or one processor-direct slot and one NuBus slot	Five NuBus slots or one processor-direct slot and four NuBus slots
Keyboard and mouse interface	Two ADB ports	One ADB port	Two ADB ports	One ADB port
Serial ports	Two mini 8-pin connectors supporting RS-422 and AppleTalk	Two mini 8-pin connectors supporting RS-232, RS-422, and AppleTalk	Two mini 8-pin connectors supporting RS-232, RS-422, and AppleTalk	Two mini 8-pin connectors supporting RS-232, RS-422, and AppleTalk
Floppy disk support	SWIM chip controls two 1.4 MB, 3.5" SuperDrives (one internal and one optional external)	SWIM chip controls one internal 1.4 MB, 3.5" SuperDrive and one optional external 800 KB or 1.4 MB disk drive	SWIM chip controls one internal 1.4 MB, 3.5" SuperDrive	SWIM chip controls one internal 1.4 MB, 3.5" SuperDrive
SCSI ports	One internal 50-pin, one external DB-25	One internal 50-pin, one external DB-25	One internal 50-pin, one external DB-25	Two internal 50-pin, one external DB-25
Sound	Apple Sound Chip	Apple Sound Chip	Enhanced Apple Sound Chip, DFAC (Digitally Filtered Audio Chip), and Sporty chip, which provide Apple Sound Chip compatibility	Enhanced Apple Sound Chip, DFAC, and Sporty chip, which provide Apple Sound Chip compatibility
Battery	Long-life lithium battery	Long-life lithium battery	Long-life lithium battery	Long-life lithium battery

Hardware architecture

The following discussion is brief and intended only to show the place of the NuBus in the computer architecture. For a complete description of hardware operation, see the *Guide to the Macintosh Family Hardware*, second edition (which supersedes the *Macintosh Family Hardware Reference*). Also useful are the *Macintosh IIsi, LC, and Classic Developer Notes* and the *Macintosh Classic II, Macintosh PowerBook Family, and Macintosh Quadra Family Developer Notes.* The *Technical Introduction to the Macintosh Family* contains a higher-level overview.

Block diagrams of the Macintosh computers that offer a NuBus expansion interface are shown in Figures 1-1 through 1-7. The Macintosh II (Figure 1-1) contains a Motorola MC68020 microprocessor driven by a 15.6672 megahertz (MHz) clock. The Macintosh IIx and the Macintosh IIcx (Figure 1-2) contain the Motorola MC68030 microprocessor, which is also driven by a 15.6672 MHz clock. The Macintosh IIci (Figure 1-3), the Macintosh IIfx (Figure 1-4), and the Macintosh IIsi (Figure 1-5) use the MC68030 microprocessor as well, but the Macintosh IIsi is driven by a 20 MHz clock, the Macintosh IIci is driven by a 25 MHz clock, and the Macintosh IIfx is driven by a 40 MHz clock. The two latest computers to offer a NuBus expansion interface include the Macintosh Quadra 700 (Figure 1-6) and the Macintosh Quadra 900 (Figure 1-7). Both use the more powerful MC68040 microprocessor, running at 25 MHz.

All Macintosh computers that provide a NuBus expansion interface use similar integrated circuits (ICs) that enable the microprocessor to communicate with external devices. These ICs are shown in the block diagrams of Figures 1-1 through 1-7 and include

- a Versatile Interface Adapter (VIA1) for communicating with the ADB transceiver, which, in turn, communicates with the mouse and keyboard.
- another Versatile Interface Adapter (VIA2) for handling interrupts from the NuBus slots (used only in the Macintosh II, Macintosh IIx, Macintosh IIcx, Macintosh Quadra 700, and Macintosh Quadra 900).
- a SCSI (Small Computer System Interface) chip for high-speed data transfer with the internal hard disk and any other SCSI device (integrated into the SCSI DMA chip on the Macintosh IIfx). The SCSI chip is used in all Macintosh II models except the Macintosh IIsi. On machines that use it, the SCSI chip is a 53C80 chip or derivative; however, the Macintosh Quadra 700 and the Macintosh Quadra 900 use a 53C96.
- a Serial Communications Controller (SCC) for serial communications. In the Macintosh IIsi, a new custom chip, Combo, combines the functions of the SCC and the SCSI controller in a single device. This Combo chip includes a 53C80 device and is completely software compatible with the SCC and SCSI chips it replaces.

- an Apple custom chip, called the IWM (Integrated Woz Machine), for controlling 800 KB, 3.5-inch floppy disk drives in the Macintosh II; another Apple custom chip, called the SWIM (Super Woz Integrated Machine), replaces the IWM chip in the Macintosh IIx, Macintosh IIcx, Macintosh IIci, Macintosh IIsi, Macintosh IIfx, Macintosh Quadra 700, and Macintosh Quadra 900. SWIM controls the 1.4 MB, 3.5-inch SuperDrives as well as 800 KB floppy disk drives.
- the Apple Sound Chip (ASC) sound generator; another Apple custom chip, the Enhanced Apple Sound chip, replaces the ASC in the Macintosh Quadra 700 and the Macintosh Quadra 900. The Enhanced Apple Sound chip continues to provide ASC compatibility.

In addition, the Macintosh IIfx includes several Apple custom ICs that enable the microprocessor to communicate with external devices. These ICs are shown in Figure 1-4 and include

- a SCSI DMA chip, which not only provides all of the functions of the SCSI chip used in the other Macintosh II computers, but can also transfer data to and from the main processor by direct-memory access (DMA) (The DMA capability is not supported by current system software.)
- an SCC IOP (I/O processor) chip that provides an intelligent interface to the Serial Communications Controller
- a SWIM-ADB IOP chip that provides an intelligent interface to the SWIM-ADB controller
- an FMC (fast memory controller) that supports high-speed cache memory, main RAM, and ROM
- an OSS (Operating System Support) chip that handles interrupts and device decodes, functions that are performed by the VIAs and the GLU chips in the other Macintosh computers

In earlier Macintosh II computers, a NuBus expansion card was the only way to provide video display. The Macintosh IIci and the Macintosh IIsi also provide the option of using a NuBus video card, but their primary video interface is built into the computers. Video signals are generated by the Apple custom RBV (RAM-based video) chip and are driven through the VDAC (video digital-to-analog converter) and CLUT (color look-up table).

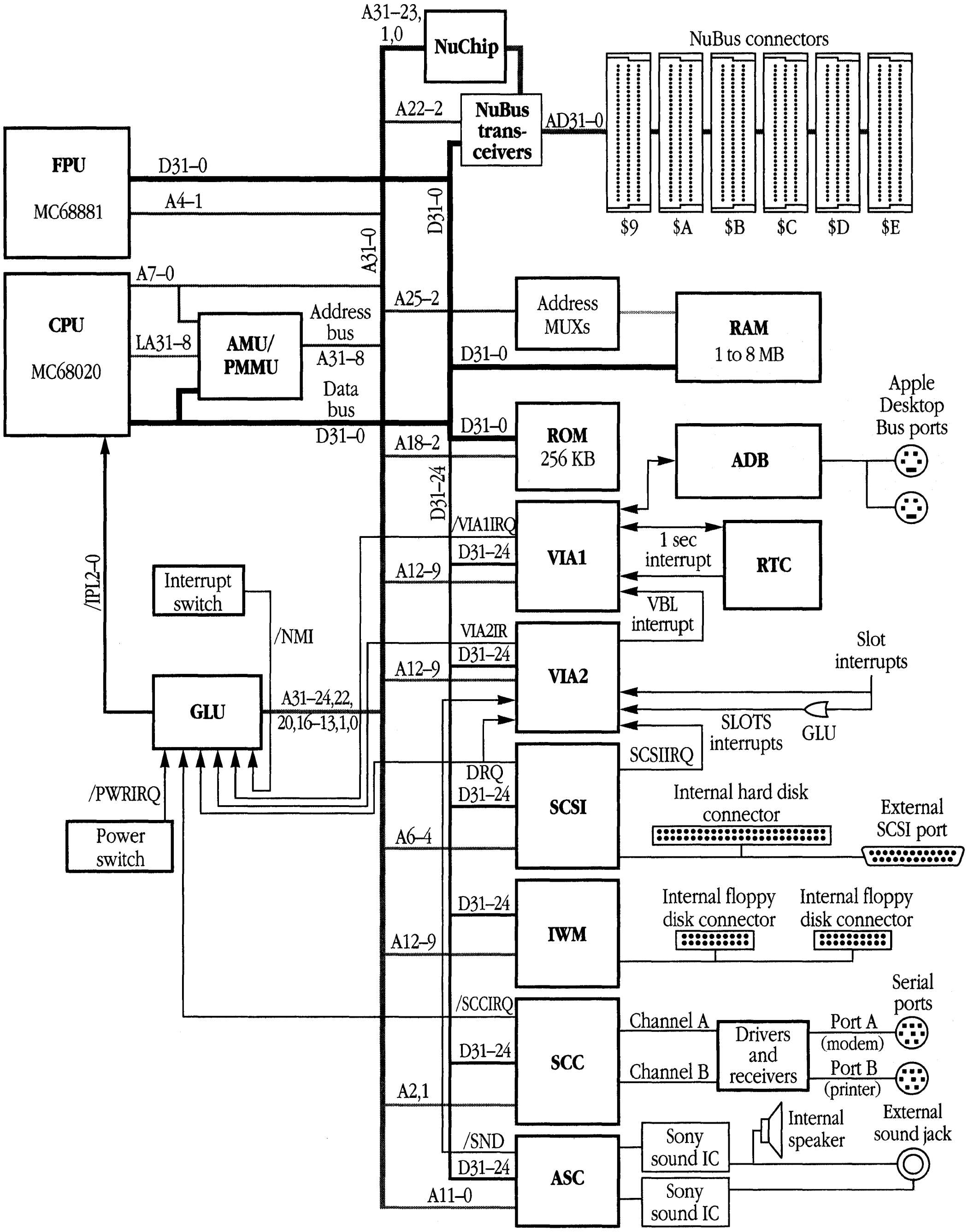
The Macintosh IIsi computer uses a microcontroller, which integrates the functions of the ADB (Apple Desktop Bus) controller, RTC (real-time clock), PRAM (parameter RAM), soft power control, power-on reset, keyboard reset, and NMI (nonmaskable interrupt). In older Macintosh models, these functions were provided by separate devices on the main logic board. Some new functions supported by the ADB microcontroller include programmable wake-up and file server mode.

The floating-point numeric coprocessors (MC68881 for the Macintosh II and MC68882 for the other Macintosh computers) use the coprocessor interfaces of their respective microprocessors. Note that a floating-point coprocessor is not included with the Macintosh IIsi as a standard feature; however, the FPU is available on the 68030 Direct Slot and NuBus adapter cards. The floating-point units in the Macintosh Quadra family are integrated into the MC68040 main processor. There are some differences between the floating-point unit in the MC68040 and the MC68881 and MC68882 FPUs used with the 68020-based and 68030-based Macintosh computers. However, the ROM software in the Macintosh Quadra 700 and the Macintosh Quadra 900 includes new code to support these differences.

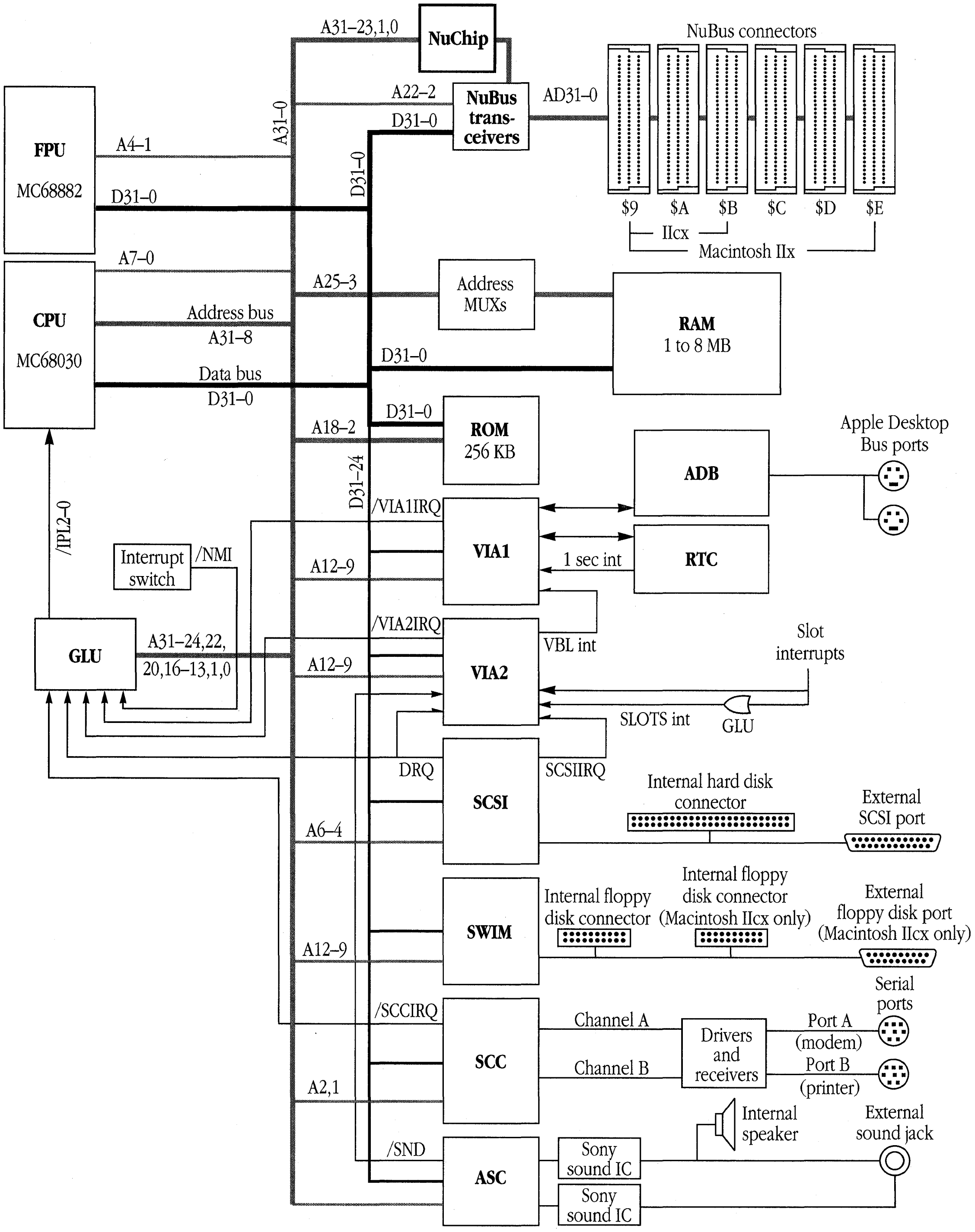
In the Macintosh Quadra family, many new custom ICs have been introduced with their new architecture. These ICs are shown in the block diagrams of the Macintosh Quadra 700 (Figure 1-6) and the Macintosh Quadra 900 (Figure 1-7).

- Along with the Enhanced Apple Sound chip, which replaces the ASC, the DFAC (Digitally Filtered Audio Chip) and the Sporty chip provide the sound interface. DFAC is a custom analog chip that provides most of the sound input functions, and the Sporty chip is a custom analog chip that provides sound output functions.
- The Caboose is a custom processor that manages the keyswitch, system power, the real-time clock, and PRAM (parameter RAM). The Caboose is used only in the Macintosh Quadra 900.
- The Direct Access Frame Buffer (DAFB) is an integrated circuit that connects directly to the system bus and controls the video RAM–based frame buffer.
- The Memory Control Unit (MCU) is a custom integrated circuit that connects to the system bus and controls access to ROM. The MCU also supports the 68040 processor's burst-mode data transfers.
- The Relayer and the Junction Data Bus (JDB) chips are two integrated circuits that make up the I/O adapter, connecting the data and control signals from the system bus and the I/O bus.
- The custom integrated circuit YANCC (Yet Another NuBus Controller Chip) controls the NuBus interface.
- Sonic is the Ethernet controller.

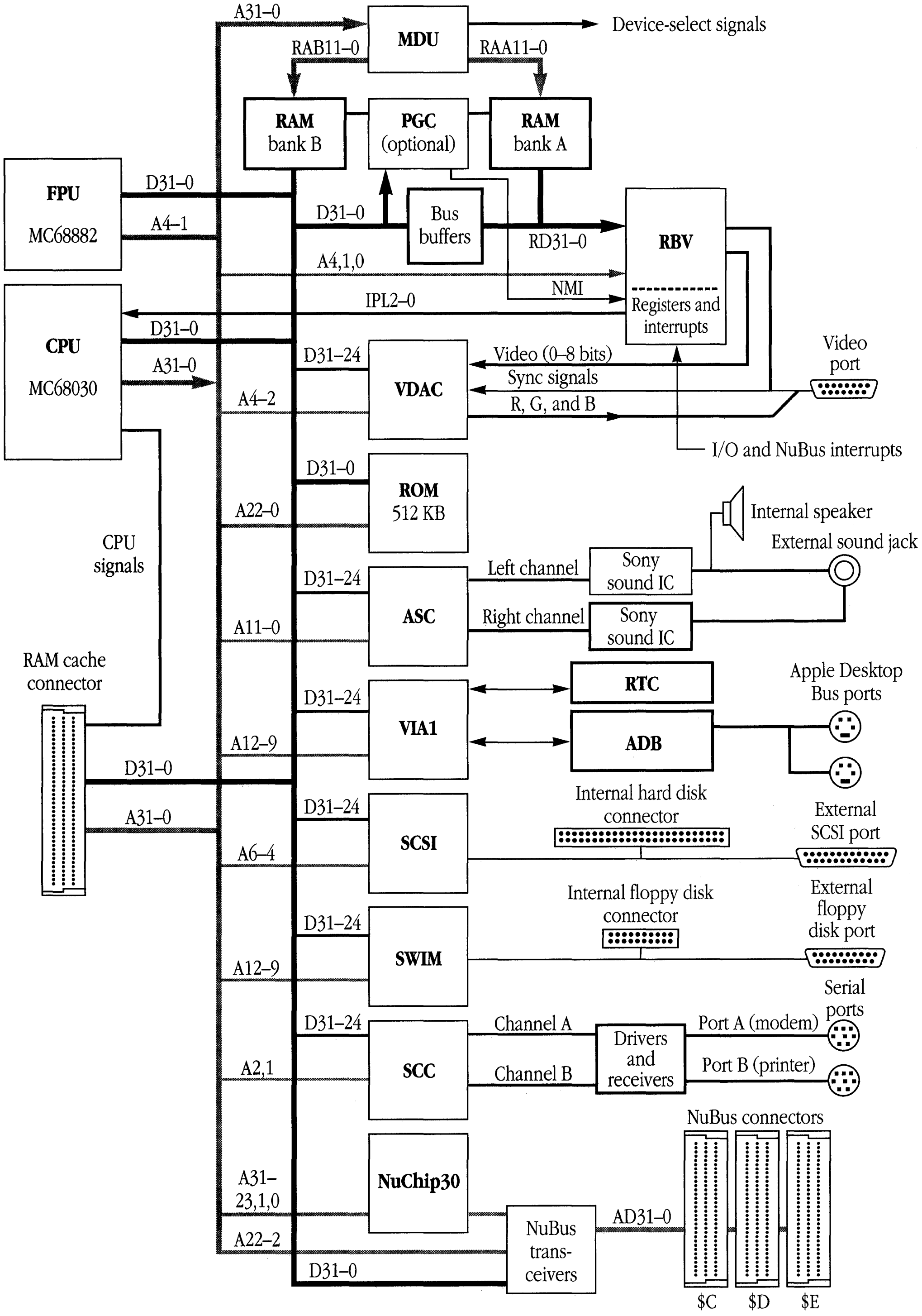
■ **Figure 1-1** Block diagram of the Macintosh II computer



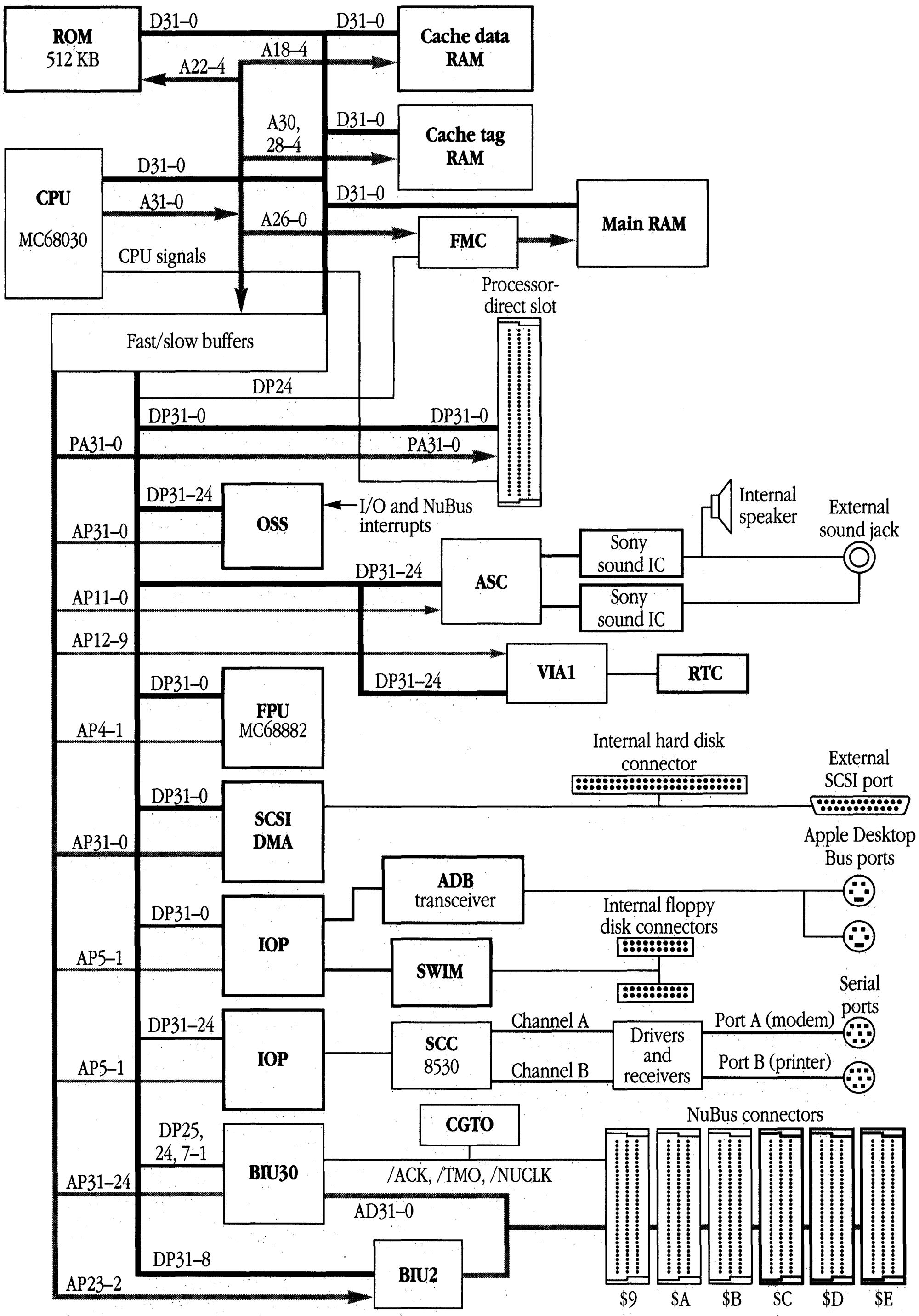
■ **Figure 1-2** Block diagram of the Macintosh IIx and Macintosh IIcx computers



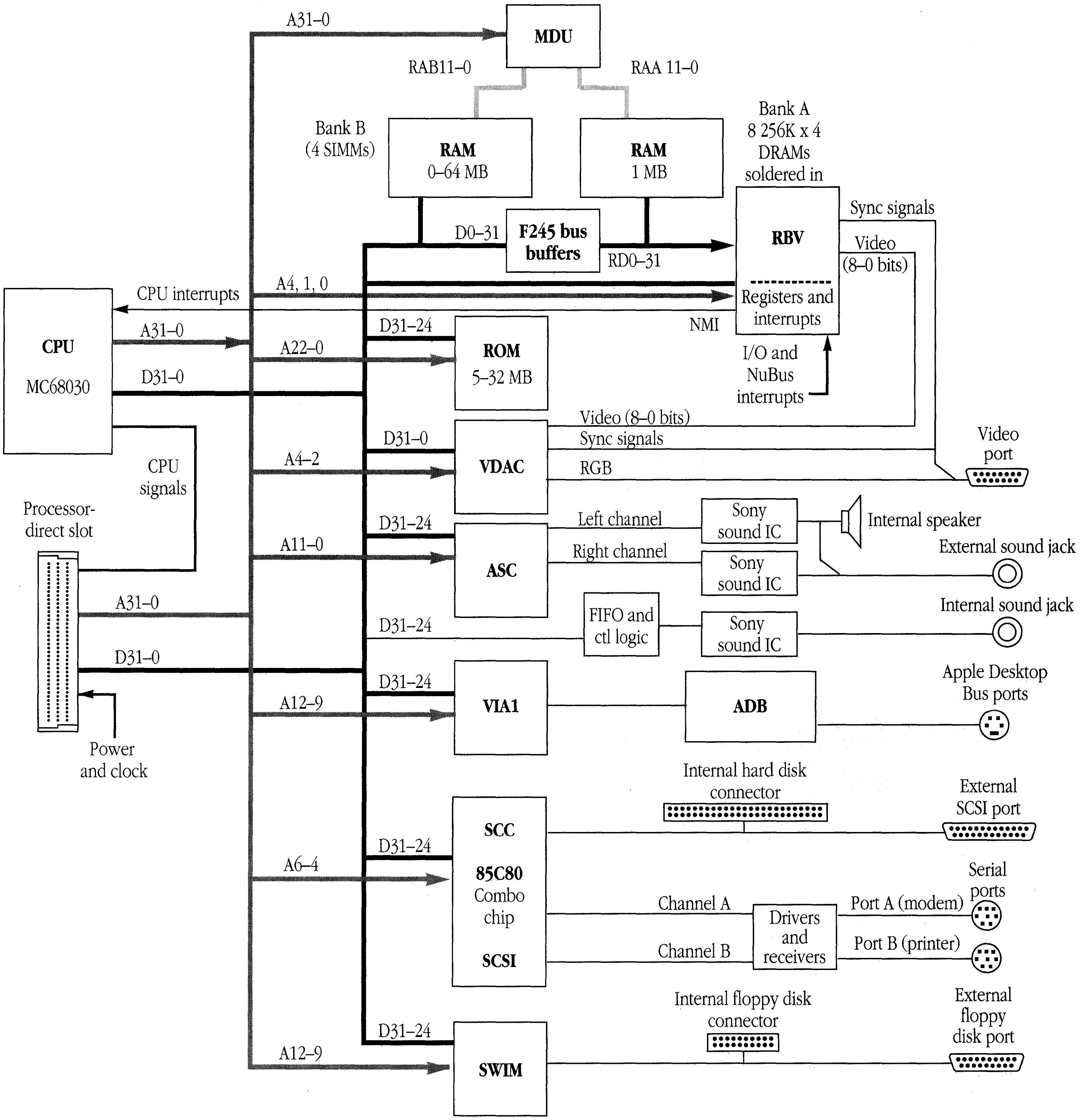
■ **Figure 1-3** Block diagram of the Macintosh IIci computer



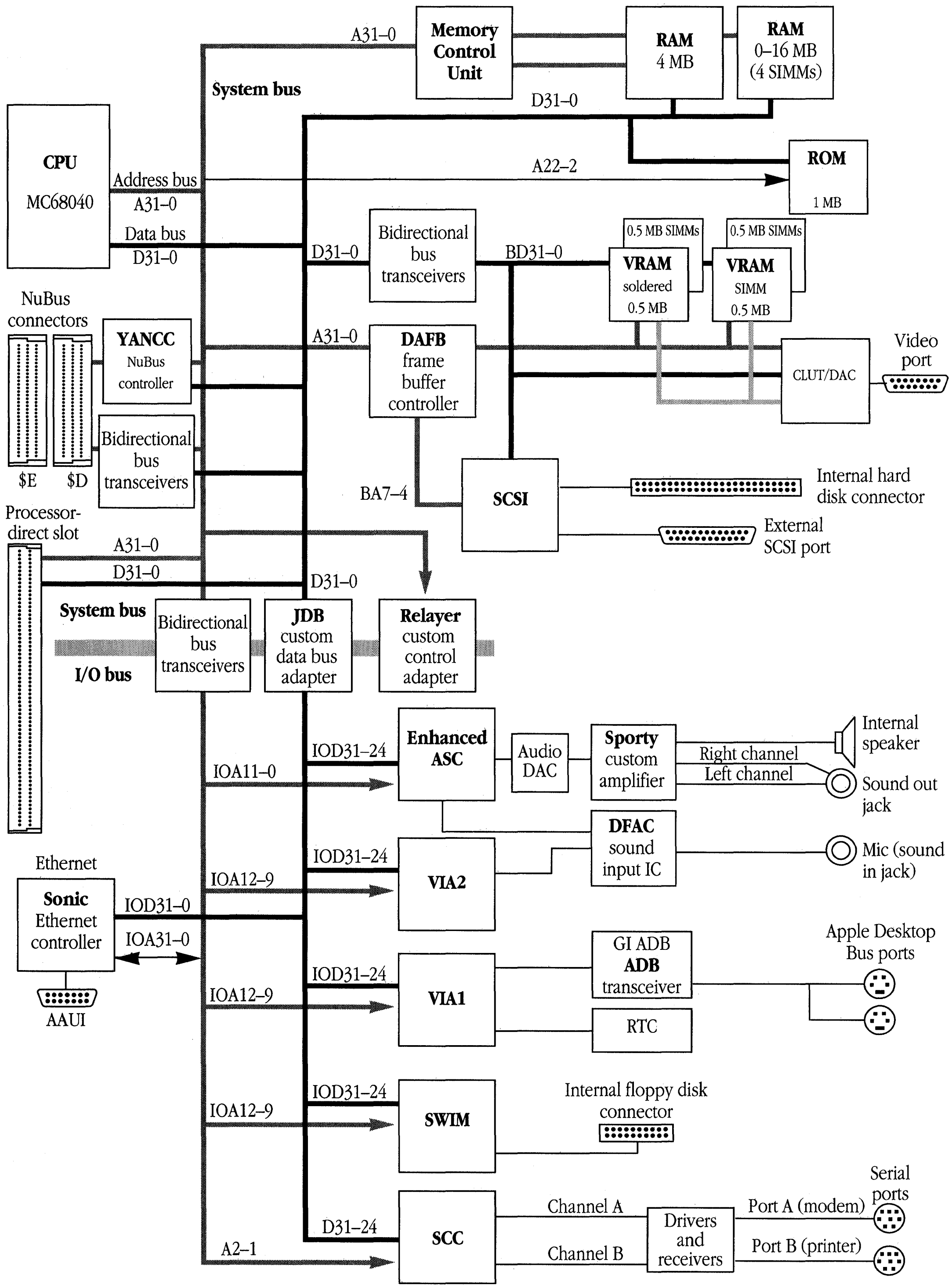
■ **Figure 1-4** Block diagram of the Macintosh IIfx computer



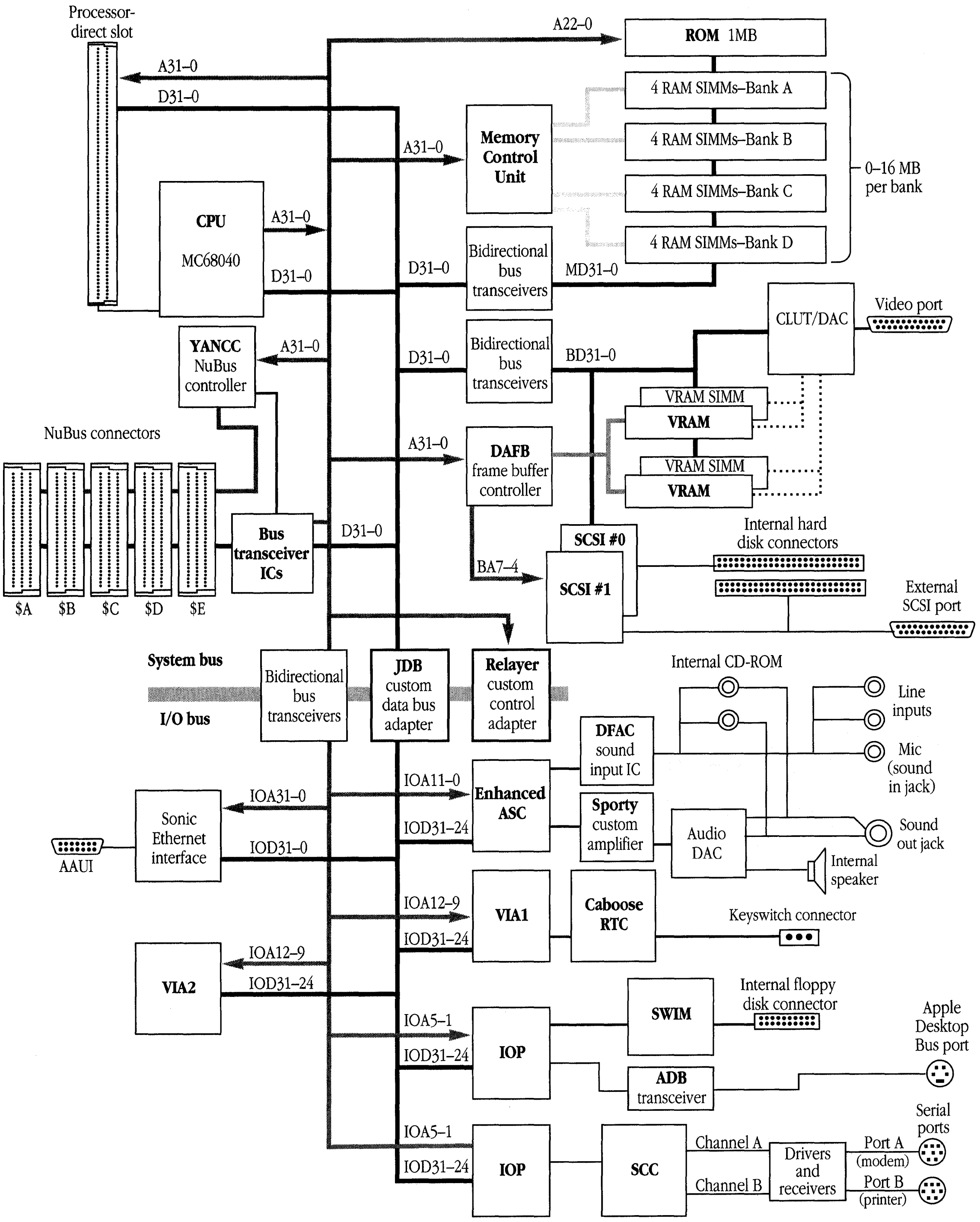
■ **Figure 1-5** Block diagram of the Macintosh IIsi computer



■ **Figure 1-6** Block diagram of the Macintosh Quadra 700 computer



■ **Figure 1-7** Block diagram of the Macintosh Quadra 900 computer



RAM

RAM is the working memory of the system. Its base address is $00. The first 1024 bytes of RAM (addresses $00 through $3FF) are normally used by the microprocessor as exception vectors; these are the addresses of the routines that gain control whenever an exception such as an interrupt or a trap occurs. RAM also contains the system and application heaps, the stack, and other information used by applications.

On most of the Macintosh computers, the microprocessor's accesses to RAM are *not* interleaved (alternated) with the video display's accesses during the active portion of a screen scan line. On these computers, the screen buffer is located in video RAM on the main logic board or a separate NuBus video card. However, on the Macintosh IIci and the Macintosh IIsi, the microprocessor's accesses to RAM are interleaved with the video display's accesses. The screen buffer for these two computers is located in main memory.

Video RAM can be located on the video card in a NuBus slot for most Macintosh computers. The Macintosh Quadra 700 and the Macintosh Quadra 900 have video RAM located on the main logic board. In the Macintosh Quadra family of computers, there can be up to four banks of video RAM. These computers also support VRAM-based NuBus cards.

In the Macintosh IIci and the Macintosh IIsi, physical memory is not in one contiguous block, so the MMU of the 68030 joins the blocks of physical memory to present contiguous logical memory to application software. The address space is decoded by the same MDU custom chip found in the Macintosh IIci; the MDU also supports burst-read mode directly to the 68030.

The physical memory in the Macintosh Quadra family of computers is set up to be contiguous. The Macintosh Quadra 700 supports up to two banks of memory, and the Macintosh Quadra 900 supports up to four banks of memory. When the Macintosh Quadra–family computer is powered on, the start code in the ROM physically "stitches" the memory together so that all the bank addresses are contiguous. The Memory Control Unit in the Macintosh Quadra 700 and the Macintosh Quadra 900 supports all 68040 memory-access types, including burst reads and burst writes.

ROM

ROM is the system's permanent read-only memory. Its base address is available in the global variable `ROMBase`. ROM contains the routines for the User Interface Toolbox and Macintosh Operating System, and the various system traps.

The Macintosh Quadra 700 and Macintosh Quadra 900 are the first Macintosh computers to have 1 MB of ROM installed on the main logic board. The ROM software for the Macintosh Quadra family of computers is the latest in a series of ROM designs and is derived from the ROM for the Macintosh IIci, Macintosh IIfx, Macintosh IIsi, and Macintosh LC computers.

Device I/O

Computers in the Macintosh family use memory-mapped I/O, which means that each device in the system is accessed by reading or writing to specific locations in the address space of the computer. The address space reserved for device I/O contains blocks devoted to each of the devices within the computer. Each device contains logic that recognizes when it's being accessed, and the device responds in the appropriate manner.

For compatibility with MC68000-based Macintosh computers, the Macintosh Operating System operates by default with 24-bit addressing. In System 7, however, you can choose to run with 32-bit addressing. New applications can take advantage of the full 32-bit addressing mode for slot access, as explained in Chapter 7, "NuBus Card Memory Access." You must not assume the system software is operating in a specific mode. Please refer to the compatibility guidelines information in *Inside Macintosh* for more information about the addressing mode selector.

Separate address spaces are reserved for processor access to cards in NuBus slots. For a device in NuBus slot number *s,* the address space in 32-bit mode begins at address $Fs00 0000 and continues through the highest address, $FsFF FFFF (where *s* is a constant in the range $9 through $E for the Macintosh II, the Macintosh IIx, and the Macintosh IIfx; $A through $E for the Macintosh Quadra 900; $9 through $B for the Macintosh IIcx; $C through $E for the Macintosh IIci; $D and $E for the Macintosh Quadra 700; and $9 for the Macintosh IIsi).

The microprocessor can directly access 2^{32} bytes, or 4 GB, of address space. In a Macintosh computer, this address space is partially accessible when the Macintosh Operating System is in 24-bit mode and totally accessible when it is in 32-bit mode.

A driver must switch into the machine's boot mode (either 24-bit or 32-bit mode) before calling ROM routines. If the operating system has been booted in 24-bit mode, an application or a driver may switch to 32-bit mode but must switch back to 24-bit mode before calling most ROM routines. More information about the addressing mode selector can found in the compatibility guidelines information in *Inside Macintosh.*

If an application needs access to a NuBus card in 32-bit mode (because it needs access to more than 1 MB of slot space, for example), it can use the system call `SwapMMUMode` to perform mode switching. Even if a machine is booted with 32-bit addressing, it is usually still necessary to call `SwapMMUMode` to ensure that the machine is in 32-bit mode. It is not correct to assume that a machine must boot with 32-bit addressing in order to work with a given driver. The `SwapMMUMode` call is described in the discussion on operating-system utilities in *Inside Macintosh.*

Memory management in the Macintosh II is provided by the Address Management Unit. The main function of the AMU is to accomplish a 24-to-32-bit memory mapping translation. A bit in VIA2 controls the mode change. This method offers the direct use of all 32 bits in one mode and a mapped set of addresses in 24-bit mode. You must replace the AMU with the MC68851 Paged Memory Management Unit (PMMU) if you are running virtual operating systems such as A/UX because A/UX runs entirely in 32-bit mode. The MC68851 PMMU is also capable of ignoring the high 8 bits of the address in order for the Macintosh Operating System to run in 24-bit mode.

Macintosh computers that use the MC68030 and MC68040 microprocessors do not need an AMU or a PMMU. The microprocessors in these computers include a built-in memory management unit that provides all necessary memory management functions, including 24-to-32-bit memory mapping translation and A/UX operating-system support.

Chapter 7, "NuBus Card Memory Access," shows in detail how cards installed in NuBus slots address memory.

Address/data bus

The block diagrams in Figures 1-1 through 1-7 show the basic address/data bus architecture used in the Macintosh computers. Note that the address and data buses are separate on the microprocessor side of the NuBus interface (transceivers and control), and that the addresses and data are multiplexed on the NuBus side of the interface. The NuBus utility, control, arbitration, and slot ID signals are described in detail in Chapter 2.

The 32-bit-wide multiplexed address or data bus connects to the NuBus slot connectors. See the section "Address/Data Signals" in Chapter 3 for a description of the address/data bus.

Macintosh IIsi NuBus interface

The Macintosh IIsi is different from all other Macintosh computers in its expansion interface. The Macintosh IIsi offers a NuBus expansion interface and a processor-direct slot (PDS) expansion interface. However, to install either type of expansion card, you must first install a special adapter card for the Macintosh IIsi.

There is a single 120-pin expansion connector on the main logic board. You can install a NuBus adapter card or a 68030 Direct Slot adapter card in the expansion connector. Both NuBus and PDS adapter cards include 68882 floating-point units for numeric coprocessing. If customers want numeric coprocessing but don't care about expansion, they still have to use an adapter card. Both adapter cards are user installable.

A NuBus adapter kit, available from an authorized Apple dealer, allows a customer to install a NuBus card in the Macintosh IIsi computer and have it function exactly as if it were in any other Macintosh computer with a NuBus expansion. Information about the physical and electrical implementation of the NuBus adapter card for the Macintosh IIsi can be found in Chapters 5 and 6.

NuBus interface architecture

All Macintosh computers that offer the NuBus expansion use a similar interface architecture to communicate with the NuBus. The most noticeable difference is that as Macintosh computers have evolved, they have incorporated different NuBus control and data transceivers into their bus interface design. This section shows how the processor uses the bus interface control and transceiver logic to communicate with the NuBus.

The Macintosh II, the Macintosh IIx, and the Macintosh IIcx computers all use the NuChip custom IC for their bus interface control function, as shown in Figures 1-1 and 1-2. In the Macintosh IIci (Figure 1-3) and the Macintosh IIsi adapter card, the bus interface control function is replaced by a NuChip 30 custom IC. In the Macintosh IIfx (Figure 1-4), two custom ICs, the BIU30 and the BIU2, perform the control and transceiver functions.

The Macintosh Quadra 700 (Figure 1-6) and the Macintosh Quadra 900 (Figure 1-7) use three chips for the interface between the system bus and the NuBus: the YANCC (Yet Another NuBus Controller Chip) IC and two 16-bit transceiver ICs. The transceiver ICs are the same type as the ones used in the Macintosh IIci. The features of the YANCC include

- support for all types of single data transfers in either direction
- a 32-bit buffer for pended writes from the MC68040 to the NuBus
- support for block move transfers between NuBus masters and main memory
- support for pseudoblock transfers between the MC68040 and NuBus slaves
- support for some new functions defined in the latest NuBus specification (refer to Chapter 2 for more information about these new features)

Unlike the NuBus controllers in previous Macintosh computers, the YANCC generates an interrupt when there is an error involving the write buffer. Software controls this interrupt by means of a control and status register in the YANCC.

The NuBus interface in Figures 1-1 through 1-7 shows bidirectional bus interface blocks between the microprocessor bus and the NuBus. Figure 1-8 shows a further breakdown of the functional elements comprising the bus interface circuits.

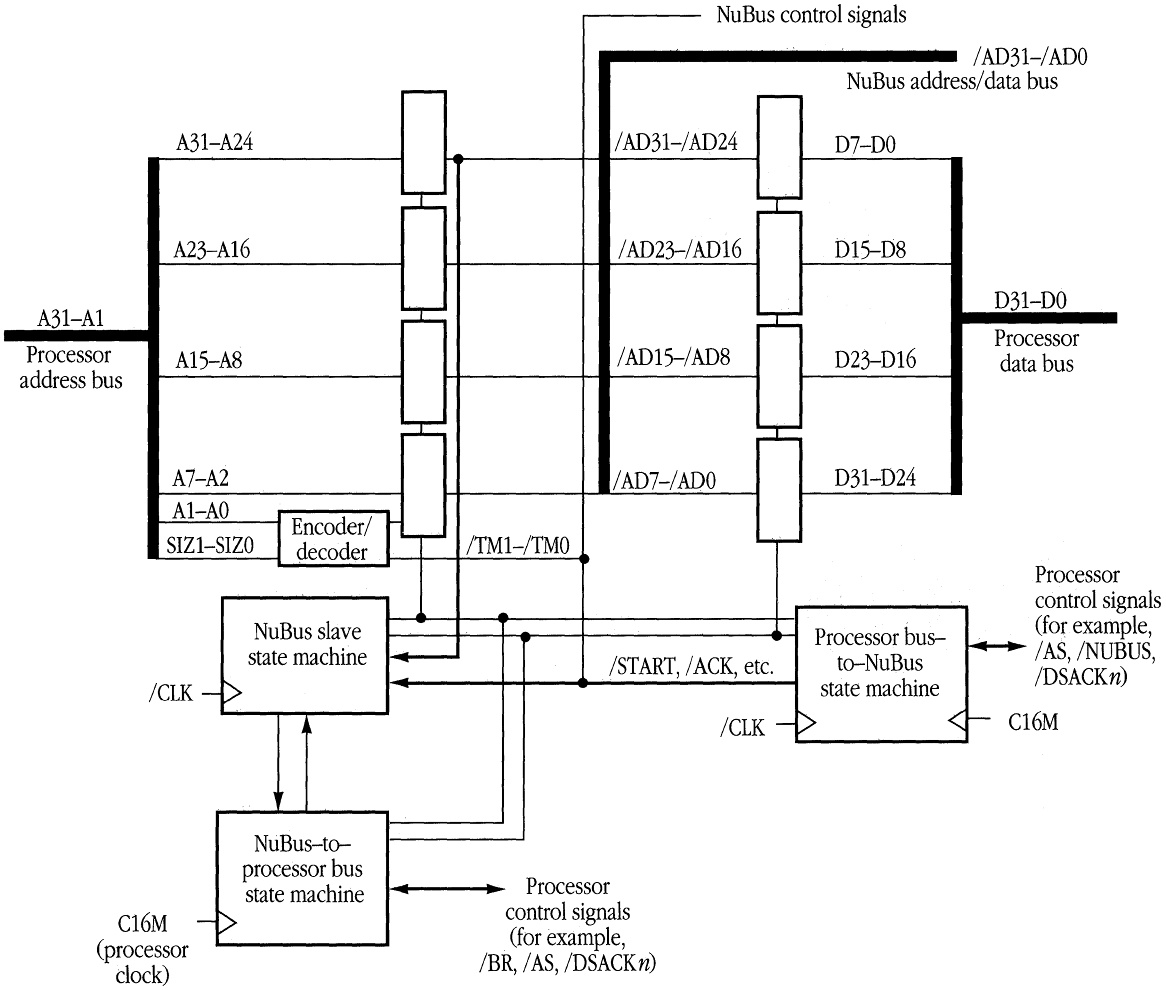
Figure 1-8 shows the bus interface architecture implemented in the Macintosh II, the Macintosh IIx, and the Macintosh IIcx. Although other Macintosh machines may vary from this design, the figure is meant primarily as an example. The bus interface control function is implemented as four state machines, three of which are shown in Figure 1-8. The fourth state machine prevents the NuBus from indefinitely awaiting an acknowledge by generating an acknowledge cycle in response to /START after 256 bus cycles (25.6 μs). A wait this long occurs when the processor makes an access to nonfunctional addresses, perhaps because the card being addressed is not present in any of the NuBus slots.

Processor bus–to–NuBus state machine

The processor bus–to–NuBus state machine (see Figure 1-8) is activated whenever the microprocessor generates a physical address from $6000 0000 through $FFFF FFFF in the data or program address spaces (see the memory map, Figure 1-9). The state machine synchronizes the request with the NuBus clock and presents the same address over the NuBus. If a slave device on NuBus responds, the data is transferred. If no slave responds, a NuBus time-out occurs and a bus error (/BERR) signal is sent to the processor. The processor can then determine the cause of the error.

◆ *Note:* A special check is made for access to $F0xx xxxx, which is the main logic board's slot address; if attempted, a bus error signal is generated immediately and no NuBus transaction is attempted.

■ **Figure 1-8** Bus interface architecture for the Macintosh II, Macintosh IIx, and Macintosh IIcx computers



NuBus–to–processor bus state machines

Two state machines, the NuBus slave and the NuBus to processor bus, control a NuBus–to–processor bus data transfer (see Figure 1-8).

The NuBus–to–processor bus state machine controls accesses from the NuBus, through the processor bus, to RAM, ROM, and I/O. For example, if an address from $0000 0000 through $3FFF FFFF is presented on the NuBus, then the NuBus–to–processor bus state machine requests the processor bus from the microprocessor and performs a RAM access to the same address. Similarly, if an access in address space $F080 0000 through $F0FF FFFF is made on the NuBus, an access in $4x00 0000 through $4xFF FFFF on the processor bus is made to the ROM (see the map in Figure 1-9). Chapter 7 provides much more detailed information on memory access. See Table 7-2 in particular.

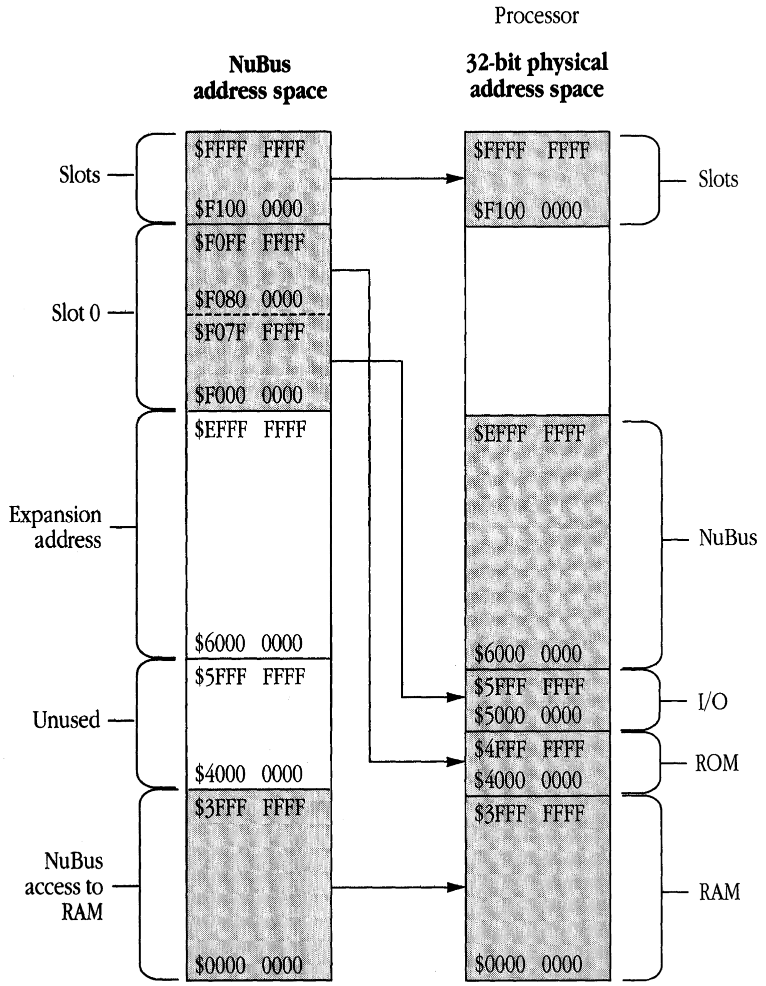
▲ **Warning** The ability to access processor bus I/O devices is not intended for normal use. Access to anything other than ROM or RAM will probably not be supported on future systems. ▲

The Macintosh II, the Macintosh IIx, and the Macintosh IIcx have only 256 KB of ROM; in these machines, only 18 bits of addressing is required to specify a ROM location. The hardware decode logic interprets any physical address whose upper 4 address bits (A31–A28) are equal to $4 as a ROM access. So there are 32 minus 4, or 28, bits available to access the locations in a ROM that requires only 18 bits of addressing. This means that 10 address bits are "don't cares" and that on the map of physical addresses there are 1024 (2^{10}) different addresses (aliases) that will access the same ROM location. (In some memory architectures, not all of these aliases are accessible from the main processor.) This act of gaining access to a memory location from several different addresses is called **aliasing.** It usually occurs in computer systems when an incomplete address decoding mechanism is used.

The NuBus–to–processor bus state machine also monitors and records when the NuBus master initiates an attention-resource-lock cycle and controls the subsequent events of a resource-locked transaction, as described in the section "Bus Locking" in Chapter 4.

The NuBus slave state machine is synchronous to the NuBus and tracks the state changes on the NuBus.

■ **Figure 1-9** NuBus–to–processor bus translation



Chapter 2 **NuBus Overview**

This chapter describes NuBus features, provides a simplified diagram of the NuBus hardware, defines many NuBus terms, classifies the signals used to implement communication over the bus, and discusses the most basic timing and transaction cycle relationships.

NuBus features

The NuBus is used for expansion of a modular Macintosh computer beyond the capabilities of the ports (connectors) on the back of the machine.

NuBus is a 32-bit-wide bus chosen by Apple to mechanize the multislot expansion of the Macintosh computers. Table 2-1 shows the highest-level design objectives and the supporting features of the NuBus. Apple chose the NuBus over competitors because it offered cost-effective high performance along with maturity of hardware design and production.

The latest Macintosh computers with NuBus expansion interfaces, the Macintosh Quadra 700 and the Macintosh Quadra 900, include enhancements to the NuBus implementation, such as block-transfer modes and new clock and serial bus signals. These changes correspond to the latest 1990 draft specification of NuBus, the *Standard for a Simple 32-Bit Backplane Bus: NuBus,* ANSI/IEEE Std 1196-1990.

■ **Table 2-1** Design objectives and features

Design objective	Supporting features
System architecture independent	Optimized for 32-bit transfers, but supports 8-bit and 16-bit nonjustified transfers. Not based on the control structure of a particular microprocessor.
High-speed data transfer	10 MHz clock synchronizes bus arbitration and transfers of read/write data to a single 32-bit address space (block transfers are currently implemented only on the Macintosh Quadra family of computers).
Simplicity of protocol	Reads and writes are the only operations used. I/O and interrupts are memory mapped. Single, large physical address space allows uniform access to all addressable cards or other resources.
Small pin count	Multiplexed data and address lines. Simplified connection, only 51 signals plus power and ground lines.
Ease of system configuration	Geographical addressing (ID lines) enables interface system to be free of DIP switches and jumpers. Distributed, parallel arbitration eliminates jumper wiring of slots with missing cards (daisy-chaining).

NuBus elements

The NuBus is a synchronous bus; all transitions and signal samplings are synchronized to a central system clock. However, it has many of the features of an asynchronous bus; transactions may be a variable number of clock periods long. This design provides the adaptability of an asynchronous bus with the design simplicity of a synchronous bus.

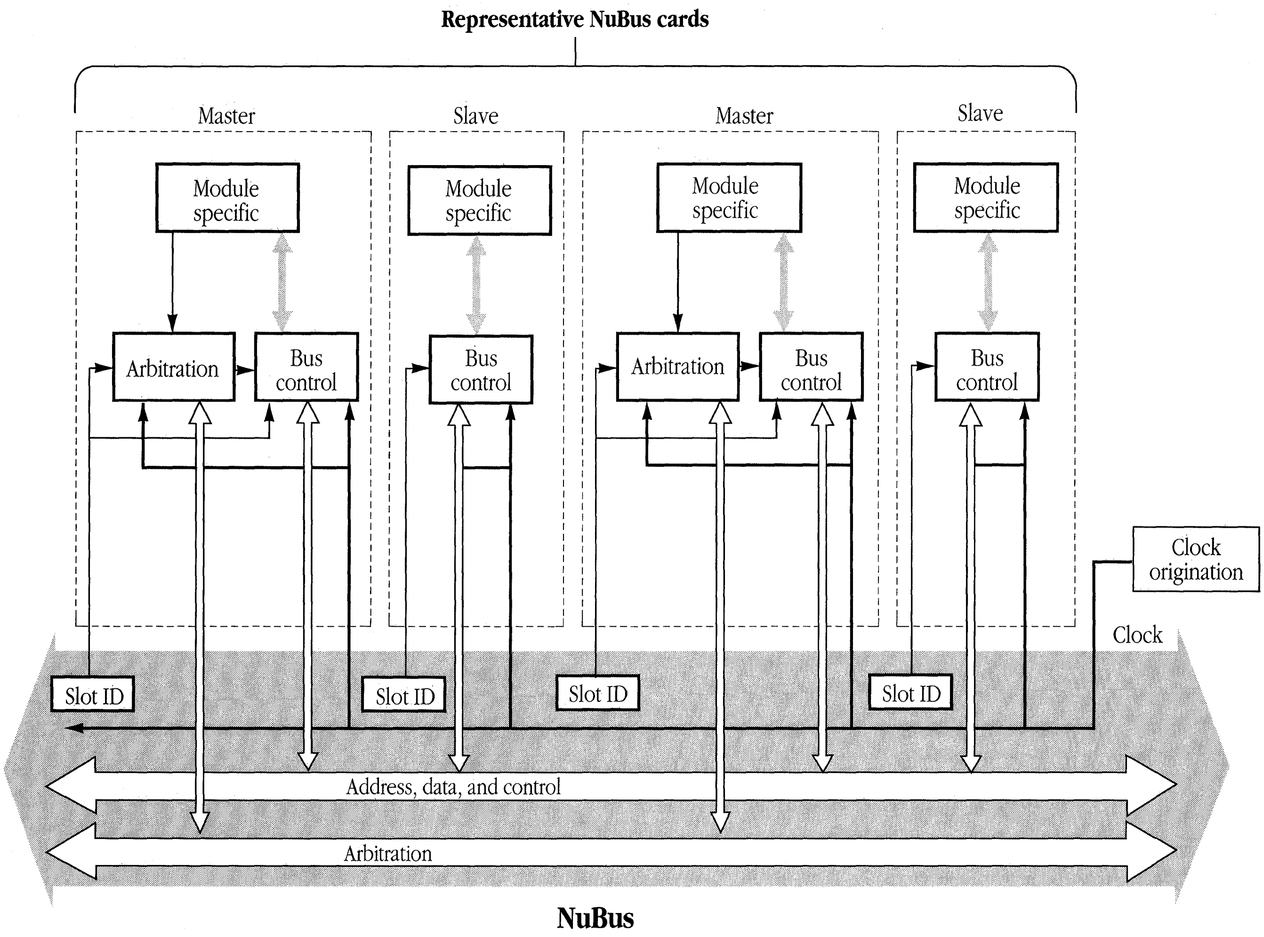
Figure 2-1 is a simplified representation of a typical NuBus system. Keep in mind that the number of NuBus cards varies for each Macintosh computer: up to six in the Macintosh II, IIx, and IIfx; five in the Macintosh Quadra 90; three in the Macintosh IIcx and IIci; and two in the Macintosh Quadra 700. In addition to the slot identification (ID), clock, address/data, and arbitration lines shown in the diagram, there are system reset, parity, power fail warning, nonmaster request, and data-transfer control lines.

NuBus supports only read and write operations in a single address space, in contrast to some other bus designs. I/O and interrupts may be accomplished within these read and write mechanisms. In the Macintosh computers with NuBus, however, interrupts are detected through the nonmaster request line (see "Interrupt Operations" in Chapter 3).

The cards in NuBus slots are peers; no card or slot is a default master. The exception is that only one card drives the system clock line; the clock is supplied by the main logic board. Each slot has an ID code hard-wired into the main logic board of the computer. This allows cards to differentiate themselves without the computer user having to arrange jumpers or adjust DIP switches.

The NuBus supports multiprocessing and other sophisticated system architectures with a few simple mechanisms explained in Chapter 3, "NuBus Data Transfer."

■ **Figure 2-1** Simplified NuBus diagram



NuBus '90 features

NuBus '90 is the 1990 proposal of the IEEE standard for the NuBus (*Standard for a Simple 32-Bit Backplane Bus: NuBus,* ANSI/IEEE Std 1196-1990). Two of the latest Macintosh computers, the Macintosh Quadra 700 and the Macintosh Quadra 900, provide the following new features in that proposal:

- On the Macintosh Quadra 900, low current at +5 V is available on the new STDBYPWR pin when main power is off and the AC cord is plugged in. This signal is not available on the Macintosh Quadra 700.
- NuBus '90 defines new signals /SB0 and /SB1 for a serial bus on the formerly reserved pins A2 and C2. The serial signals are bused and terminated, but the main circuit board does not drive them.
- New signals /TM2, /CLK2X, and /CLK2XEN support block transfers at double the standard rate. The Macintosh Quadra family of computers allows double-rate block transfers between NuBus cards, but does not support double-rate transfers to or from the main memory.
- NuBus '90 defines new signals /CM0, /CM1, /CM2, and /CBUSY to support a cache-coherency protocol. Pins on the NuBus connector are assigned to those signals, but the Macintosh Quadra 700 and the Macintosh Quadra 900 do not support them.

NuBus signal classifications

There are some slight changes in the signal definitions in the original NuBus implementation and NuBus '90. Table 2-2 shows the NuBus signal classifications based on the original implementation of NuBus in Macintosh computers, and Table 2-3 shows the NuBus '90 signal classifications. In the first table, NuBus signals can be grouped into six classes based on the functions that they perform. For the NuBus '90 signal classes (Table 2-3), the signals are grouped into eight separate classes. There are also power and ground lines.

■ **Table 2-2** Signal classifications in the original NuBus implementation

Classification	Signal	Signal description	Number of pins
Address/data	/AD31–/AD0	Address/data	32
Arbitration	/ARB3–/ARB0	Arbitration	4
	/RQST	Request	1
Control	/START	Start	1
	/ACK	Acknowledge	1
	/TM0	Transfer mode 0	1
	/TM1	Transfer mode 1	1
Parity	/SP	System parity	1
	/SPV	System parity valid	1
Slot ID	/ID3–/ID0	Slot identification	4
Utility	/RESET	Reset	1
	/CLK	Clock	1
	/PFW	Power fail warning	1
	/NMRQ	Nonmaster request	1
		Total signals	**51**
Power/ground	+5V		11
	+12V		2
	–12V		2
	–5.2V (not supplied)†		8
	GND	Ground	20
	Reserved	Reserved	2
		Total pin count	**96**

† These pins are wired together but not supplied with power from the computer.

■ **Table 2-3** Classes of NuBus '90 signals

Classification	Signal	Signal description	Number of pins
Address/data	/AD31–/AD0	Address/data	32
Arbitration	/ARB3–/ARB0	Arbitration	4
	/RQST	Request	1
Cache coherency	/CBUSY	Cache busy	1
	/CM2–/CM0	Cache maintenance	3
Control	/START	Start	1
	/ACK	Acknowledge	1
	/TM0	Transfer mode 0	1
	/TM1	Transfer mode 1	1
	/TM2	Transfer mode 2	1
Parity	/SP	System parity	1
	/SPV	System parity valid	1
Serial bus	/SB1–/SB0	Serial bus signals	2
Slot ID	/ID3–/ID0	Slot identification	4
Utility	/RESET	Reset	1
	/CLK	Clock	1
	/CLK2X	Clock2X	1
	/CLK2XEN	Clock2X enable	1
	/PFW	Power fail warning	1
	/NMRQ	Nonmaster request	1
		Total signals	**60**
Power/ground	+5V		11
	+12V		2
	–12V		2
	/STDBYPWR†	Standby power	1
	GND	Ground	20
		Total pin count	**96**

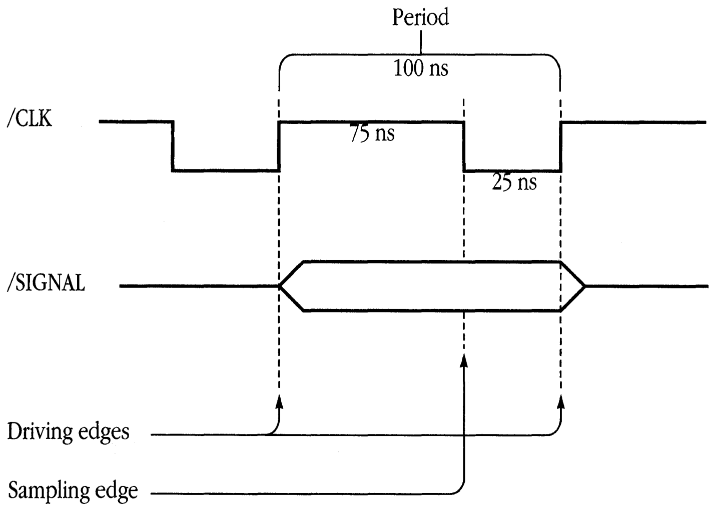
† The /STDBYPWR signal is not supported in the Macintosh Quadra 700.

△ **Important** The eight lines that were connected to the –5.2V signals in the original NuBus are now used for new features. Many older NuBus cards connect those eight lines together; the presence of such a card in the Macintosh Quadra family of computers will disable the new features of all installed NuBus cards that use those lines. All the other features of both the old and new cards will operate normally. △

NuBus timing

The NuBus system clock has a 100-nanosecond (ns) period with a 75 ns high, 25 ns low duty cycle. Figure 2-2 shows the basic timing for most NuBus signals. The low-to-high transition of /CLK is used to drive and release signals on the bus. Signals are sampled on the high-to-low transition of the clock. The asymmetric duty cycle of the clock provides 75 ns for propagation and setup time. Bus skew problems are avoided by having 25 ns between the sample and drive edges.

- **Figure 2-2** NuBus signal timing



NuBus terminology

Table 2-4 defines terms used throughout Part I that are used to describe the NuBus expansion interface. The relationships between some of these terms are illustrated in Figures 2-2 and 2-3. All NuBus signals are active (asserted) when low; a slash preceding a signal name indicates that it is active-low, for example, /START.

■ **Table 2-4** NuBus expansion interface terminology

Term	Definition
1X block transfer	A block transfer in which NuBus words are transmitted at a 10 MHz rate. These types of data transfers are only implemented on the Macintosh Quadra family of computers.
2X block transfer	A block transfer in which NuBus words are transmitted at a 20 MHz rate, sometimes referred to as *double-rate block transfers*. The Macintosh Quadra 700 and the Macintosh Quadra 900 computers allow *double-rate block transfers* between NuBus cards, but do not support them to or from the main memory.
Acknowledge (ack) cycle	Last cycle of a transaction during which /ACK is asserted by a slave responding to a master. See Figure 2-3.
Active	For an active-low signal. Synonymous with *asserted, low,* and *true.*
Arbitration contest	The mechanism used to choose which of two or more cards requesting control of the bus will become the next bus master. A complete arbitration contest requires two bus cycles (at 100 ns each).
Asserted	The logic state of an active-low signal line when the line is driven low. All NuBus signal lines are active-low. Synonymous with *active, low,* and *true.*
Attention cycle	A particular kind of start cycle, one in which both /START and /ACK are asserted. There are three types: attention-null, attention-resource-lock, and attention-cache cycles. See "Resource Locking" in Chapter 4.
Bus lock	A mechanism for providing continuing tenure (bus ownership) by a single card. The extended tenure may include multiple transactions or attention cycles. One type of attention cycle is an attention-resource-lock (often shortened to *resource lock*); therefore a bus lock may or may not include a resource lock.

(continued)

■ **Table 2-4** NuBus expansion interface terminology (continued)

Term	**Definition**
Cache coherent	The ability of a bus module to maintain a cache. The cache contains data and tags that can determine the ownership of data between multiple processors, and thus maintain coherency or agreement between data shared by processors and memory. Refer to Chapter 3 for a discussion of cache coherency. Cache coherency is supported on Macintosh computers; however, the NuBus method is not used.
Card	A printed circuit board connected to the bus in parallel with other cards.
Clock cycle	The sequence of events on the NuBus clock from one rising edge to the next, nominally 100 ns in duration and beginning at the rising edge. See Figure 2-2.
Copyback	A function in the NuBus *cache-coherency* protocol that is used to free up cache lines to service a *read miss* or *write miss* or to flush data into an I/O buffer. The copyback function may also be used during context switching.
Data cycle	Any period in which data is known to be valid and acknowledged. It includes *acknowledge cycles,* as well as intermediate data cycles within a *block transfer.* See Figure 2-3.
Deasserted	For an active-low signal. Synonymous with *high, inactive, unasserted, false,* and *released.*
Double-rate block transfer	A block transfer in which data NuBus words are transmitted at a 20 MHz rate, also referred to as *2X block transfers.* The Macintosh Quadra 700 and the Macintosh Quadra 900 computers allow *double-rate block transfers* between NuBus cards, but do not support them to or from the main memory.
Doublet	A 16-bit data item taken as a unit. Synonymous with a NuBus *halfword.*
Drive	To cause a bus signal line to be in a known, determinate state.
Driving edge	The rising edge (low to high) of the central system clock (/CLK). See Figure 2-2.
False	For an active-low signal. Synonymous with *high, inactive, deasserted, unasserted,* and *released.*

(continued)

■ **Table 2-4** NuBus expansion interface terminology (continued)

Term	**Definition**
Global	An attribute of NuBus *cache-coherency* transactions where the data is shared between multiple cache-coherent masters.
High	For an active-low signal. Synonymous with *inactive, deasserted, unasserted, false,* and *released.*
Inactive	For an active-low signal. Synonymous with *high, deasserted, unasserted, false,* and *released.*
Low	For an active-low signal. Synonymous with *active, asserted,* and *true.*
Master	A card that initiates the addressing of another card or the processor on the main logic board. The card addressed is at that time acting as a slave.
Master flow control	Used by the bus master to control the flow of data during *2X block transfers.*
M,E,S,I	Modified, exclusive, shared, invalid. The set of NuBus cache-line states and protocols used to guarantee *cache coherency* on NuBus.
Minor slot space	An Apple-specific term that describes the first megabyte of the 16 MB standard slot space. If a Macintosh computer is operating in 32-bit mode, it can access all the address space in both the standard slot and super slot spaces of any slot card. In 24-bit mode, it can address only 1 MB of each card's standard slot space. In 24-bit mode, the computer hardware translates 24-bit addresses of the form $*s*x xxxx into 32-bit addresses of the form $F*s*0x xxxx, where *s* is a digit in the range $9 through $E.
Open collector	A bus driver that drives a line low or doesn't drive it at all.
Parked	The condition when a bus master has completed a transaction and released /RQST, and before any other card has asserted /RQST. Bus parking is discussed in Chapter 4.
Period	The 100 ns duration of /CLK, the NuBus clock signal consisting of a 75 ns high state and a 25 ns low state. See Figure 2-2.

(continued)

■ **Table 2-4** NuBus expansion interface terminology (continued)

Term	Definition
Quadlet	A 32-bit data item taken as a unit. Synonymous with a NuBus *word.*
Read miss	This term is used with the NuBus *cache-coherency* protocol. When a processor attempts to read data, and the requested location within the cache is invalid, the initial read of the data in the cache fails.
Released	For an active-low signal. Synonymous with *high, inactive, deasserted, unasserted,* and *false.*
Sampling edge	The falling edge (high to low) of the central system clock (/CLK). See Figure 2-2.
Slave	A card that responds to being addressed by another card acting as a master. The main logic board in Macintosh computers may be either master or slave. Some cards may be slave-only in function because they lack the circuitry to arbitrate in a bus ownership contest.
Slave flow control	Used by the addressed slave to control the flow of data during all transactions.
Slot	A connector attached to the bus. A card may be inserted into any of the slots when more than one is provided (Macintosh II, Macintosh IIx, and Macintosh IIfx have six slots; Macintosh Quadra 900 has five slots; Macintosh IIcx and Macintosh IIci have three slots; Macintosh Quadra 700 has two slots; and Macintosh IIsi has one slot).
Slot ID	The hex number ($9 through $E in the Macintosh II, Macintosh IIx, and Macintosh IIfx; $A through $E in the Macintosh Quadra 900; $9 through $B in the Macintosh IIcx; $C through $E in the Macintosh IIci; $D through $E in the Macintosh Quadra 700; and $9 in the Macintosh IIsi) corresponding to each card slot. Each slot ID is established by the main logic board of the computer and communicated to the card through the /IDx lines.

(continued)

■ **Table 2-4** NuBus expansion interface terminology (continued)

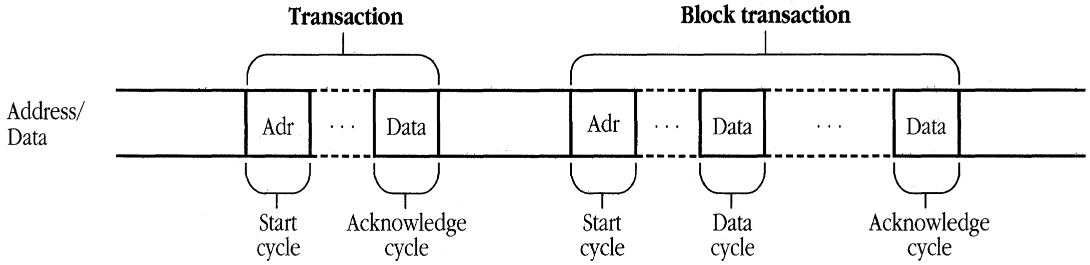
Term	Definition
Snarf	An action taken by a cache-coherent master when it eavesdrops on a *write-back* transaction and absorbs the data.
Snooping	An action taken by cache-coherent bus modules to monitor cache-coherent transactions on the bus.
Snooping module	A module that snoops a cache-coherent transaction between a master and a slave.
Standard slot space	The upper one-sixteenth of the total address space. These addresses are in the form $Fsxx xxxx, where *F, s,* and *x* are hex digits of 4 bits each and *s* represents the specific slot number. This address space is geographically divided among the NuBus slots according to slot ID numbers. Each slot space is 16 MB.
Start cycle	The first cycle of a transaction during which /START is asserted. See Figure 2-3. The start cycle is one bus clock period long; the transfer mode and the address are valid during this cycle.
Super slot space	An Apple-specific term that describes the large portion of memory in the range $9000 0000 through $EFFF FFFF. NuBus addresses of the form $*s*xxx xxxx (that is, $*s*000 0000 through $*s*FFF FFFF) address the super slot space that is assigned to the card in slot *s,* where *s* is an ID digit in the range $9 through $E. Each super slot space is 256 MB.
Tenure	A time period of unbroken bus ownership by a single master. A master may lock the bus and, during one tenure, perform several transactions. The concept of bus locking is further explained in Chapter 4 in the section "Locking."
Transaction	A complete NuBus operation such as read or write. In the Macintosh computers with NuBus, a transaction is made up of a start cycle, wait cycles as required by the responding card, and an acknowledge cycle. Start cycles are one clock period long and convey address and command information. Acknowledge cycles are also one clock period long and convey data and acknowledgment information. See Figure 2-3.

(continued)

■ **Table 2-4** NuBus expansion interface terminology (continued)

Term	Definition
Tristate	A bus driver that drives a line low or high or doesn't drive it at all. Also called three-state.
True	For an active-low signal. Synonymous with *active, asserted,* and *low.*
Unasserted	For an active-low signal. Synonymous with *high, deasserted, false, inactive,* and *released.*
Word	In Part I, *word* refers to a NuBus word *(quadlet)* and is 32 bits long; a halfword *(doublet)* is 16 bits long (usage is consistent with the Texas Instruments NuBus specification). The data type "*word,*" however, is 16 bits long; this inconsistency results from the difference between 16- and 32-bit microprocessors. Part II of this book refers to a word as 16 bits and a longword as 32 bits.
Write-back	An action taken by a *snooping* module when it returns its modified data to shared memory.
Write-back cache	Also referred to as copyback cache. A cache that does not propagate all write cycles to memory, but holds the data in the cache until the cache line holding the data must be reused.
Write miss	This term is used with the NuBus *cache-coherency* protocol. When a processor attempts to write data, and the requested location within the cache is invalid or the data is shared, the initial write to the cache fails.
Write-through cache	A cache that propagates all write cycles to memory.

■ **Figure 2-3** Cycle and transaction relationships



Chapter 3 **NuBus Data Transfer**

This chapter describes the utility signals, slot ID signals, and data-transfer signals; it then gives specifications for the process of transferring data over the NuBus interface from a master to a slave.

▲ **Warning** This chapter is intended to give the developer an overview of the specific features of NuBus. It is not intended to be used as the sole technical reference for development of a NuBus card. If you are developing a NuBus card, please use the NuBus specification as your technical reference. ▲

Utility signals

This section identifies the signal lines that serve utility functions for the NuBus interface. The main logic board of a Macintosh computer provides the structure and the slot connectors; it also provides the clock and reset signal sources and bus time-out circuitry.

Clock signals

The clock signal (/CLK), driven from a single source, synchronizes bus arbitration and data transfers between system cards. The /CLK signal has an asymmetric duty cycle of 75% high and a constant nominal frequency of 10 MHz. In general, signals are changed at the rising (driving) edge of /CLK, and they are sampled at the falling (sampling) edge.

Two additional clock signals have been defined for NuBus '90. These signals are provided only in the Macintosh Quadra 700 and the Macintosh Quadra 900. The /CLK2X signal synchronizes 2X block transfers between NuBus cards. The /CLK2X signal has a duty cycle of 50% and a constant nominal frequency of 20 MHz. During a 2X block transfer, modules drive new data on the assertion edge of /CLK2X and sample the data on the following assertion edge.

Clock2X Enable (/CLK2XEN) is a sense line to detect conflicts with the NuBus '90 signals. If there are no boards that short this line to the other NuBus '90 lines, the line stays low, enabling the /CLK2X driver.

Reset signal

The reset signal (/RESET) is an open-collector line that is asserted asynchronously to the NuBus clock. When asserted,/RESET causes a NuBus interface initialization for all cards (bus reset).

Because of the design of the computer hardware and firmware, there is a slight deviation of the duration of the /RESET signal from that specified in the IEEE 1196 NuBus standard. Durations and times are dependent on system clock frequency.

As part of the startup code in the ROM, a reset instruction is executed shortly after the microprocessor comes out of hardware reset (nominally 200 ms). The execution of this reset instruction causes a subsequent 33 µs assertion of /RESET. Thus, for each of the three ways in which reset occurs, the timing is as follows:

- **Initial power-on:** Shortly after all power supplies have stabilized, the /RESET line is driven low for a nominal 200 ms. Then, about 3 µs later and because of the startup code, /RESET is again driven low for 33 µs.
- **Pressing the reset button:** /RESET is asserted for as long as the button is held down, plus the nominal 200 ms. As in the power-on case, a subsequent 3 µs deassertion is followed by an additional 33 µs assertion.
- **Executing the restart command:** The code for this menu item executes two reset instructions separated by about 3 µs. Thus, /RESET is asserted for 33 µs, deasserted for 3 µs, and asserted again for 33 µs.

You should treat all assertions of /RESET (of any duration) identically.

Power fail warning signal

The power fail warning signal (/PFW) may be asserted asynchronous with respect to the driving edge of /CLK and indicates that the power is about to fail. In Macintosh computers, this signal is also used to control the power supply. Driving /PFW high turns the computer on; driving /PFW low turns it off.

See Chapter 5, "NuBus Card Electrical Design Guide," for /PFW drive requirements if the card you are designing is to control the power supply through the NuBus.

Nonmaster request signal

The nonmaster request signal (/NMRQ) is asynchronous to /CLK and provides an interrupt mechanism for cards that are intended to be slave-only. Such cards avoid the cost of implementing arbitration logic.

Serial bus signals

Two serial bus signals (/SB0 and /SB1) have been defined in the NuBus '90 specification. These signal lines are defined only for the Macintosh Quadra 700 and the Macintosh Quadra 900. The signal lines are bused and terminated, but the main circuit board does not drive them.

Card slot identification signals

Identification signals 3 through 0 (/ID3–/ID0) are binary coded to specify the physical location of each card. The highest-numbered slot ($F) has the four signals wired low. The lowest-numbered slot ($0) has all ID signals high. In the Macintosh II, Macintosh IIx, and Macintosh IIfx computers there are six slots numbered $9 through $E. The Macintosh Quadra 900 provides five slots numbered $A through $E. The Macintosh IIcx and Macintosh IIci have only three slots numbered $9 through $B and $C through $E, respectively. The Macintosh Quadra 700 has two slots numbered $D and $E, and the Macintosh IIsi has one slot, numbered $9. The main logic board is addressed as slot $0.

△ **Important** You must tie the ID lines high through pull-up resistors or they will not work. For example, the Macintosh II Video Card described in Chapter 11 uses 3.3 kΩ pull-up resistors; you should, however, be able to use resistors as high as 10 kΩ safely. △

The distributed arbitration logic uses the ID numbers to uniquely identify cards for arbitration contests. See Chapter 4, "NuBus Arbitration."

The ID signals are also used to allocate a small portion of the total address space to each card. The upper one-sixteenth (256 MB) of the entire 4 GB NuBus address space is called the **standard slot space.** If signals /ID3–/ID0 are used to specify NuBus address lines /AD27–/AD24, each of the 16 possible NuBus card slots has an address of the form $Fsxx xxxx, where *s* is the 4-bit hex digit for a particular slot. This address range allocates 16 MB of address space (one-sixteenth of 256 MB) per NuBus card slot, an address region called a **slot.**

However, if bits /ID3–/ID0 are used in a different way, a second natural address decode of what is called **super slot space** can be easily performed. If bits /ID3–/ID0 are used to specify NuBus address lines /AD31–/AD28, each of the 16 possible NuBus card slots has an address of the form $sxxx xxxx, where *s* is the 4-bit hex digit for a particular slot. This address range allocates 256 MB of address space (one-sixteenth of 4 GB) per NuBus card slot, an address region called a *super slot.* Thus each physical slot has allocated to it a standard slot space and a super slot space.

This fixed address allocation, based solely on the slot location of a card, enables the design of systems that are free of jumpers and switches. Chapter 7, "NuBus Card Memory Access," discusses memory addressing in detail.

Signal line determinacy

The bus driving circuitry, the bus transmission line parameters, and the terminating impedances must be coordinated to make the signal lines determinate within the specified setup and hold times of the NuBus clock.

A signal line is determinate by virtue of satisfying one of the following conditions:

- If a signal is driven during the clock cycle (or half cycle) *n,* then it is determinate during the cycle (or half cycle) *n.*
- If a signal is unasserted during cycle *n* and is not driven during cycle *n* + 1, then that signal is guaranteed to remain unasserted during cycle *n* + 1.
- If an open collector signal is driven asserted during cycle *n* and is not driven during cycle *n* + 1, then it is guaranteed to be unasserted during cycle *n* + 1.
- If a tristate signal is asserted during cycle *n* and is not driven during cycles *n* + 1 and *n* + 2, then the line is *not* guaranteed determinate during cycle *n* + 1 but is guaranteed to be unasserted during cycle *n* + 2.

Data-transfer signals

The bus data-transfer signals, including control, address/data, and bus parity, are all tristate.

Control signals

This section describes the primary functions of the four NuBus control signals.

Transfer start (/START) is driven for only one clock period by the current bus master at the beginning of a transaction. The /START signal indicates to the slaves that the address/data signals are carrying a valid address.

Transfer acknowledge (/ACK) is driven for only one clock period by the addressed slave device and indicates the completion of the transaction. An exception to the foregoing is an attention cycle, when the bus master asserts both /START and /ACK. See "Attention Cycles," later in this chapter.

Three transfer mode signals are included in the NuBus definition: /TM0, /TM1, and /TM2. The /TM2 signal was introduced in the NuBus '90 specification and is provided only in the Macintosh Quadra 700 and the Macintosh Quadra 900. The transfer mode signals are driven by the current bus master during start cycles to indicate the type of bus operation being initiated. They are also driven by bus slaves during acknowledge cycles to denote the type of acknowledgment. The /TMx encoding for start cycles is given in Table 3-1.

Address/data signals

Address/data signals 0 through 31 (/AD31–/AD0) are multiplexed to carry a 32-bit byte address at the beginning of each transaction and up to 32 bits of data later in the transaction. Note that the /AD0 and /AD1 signals, along with the /TMx lines, carry transfer mode information during the start cycle. This transfer mode encoding is shown in Table 3-1.

■ **Table 3-1** Transfer mode coding

/TM2	/TM1	/TM0	/AD1	/AD0	Type of cycle
H	L	L	L	L	Write byte 3
H	L	L	L	H	Write byte 2
H	L	L	H	L	Write byte 1
H	L	L	H	H	Write byte 0
H	L	H	L	L	Write halfword 1
H	L	H	L	H	1X block write
H	L	H	H	L	Write halfword 0
H	L	H	H	H	Write word
H	H	L	L	L	Read byte 3
H	H	L	L	H	Read byte 2
H	H	L	H	L	Read byte 1
H	H	L	H	H	Read byte 0
H	H	H	L	L	Read halfword 1
H	H	H	L	H	1X block read
H	H	H	H	L	Read halfword 0
H	H	H	H	H	Read word
L	L	H	L	H	2X block write
L	H	H	L	H	2X block read

† /TM2 is used only in the Macintosh Quadra 700 and the Macintosh Quadra 900 and is not defined for other Macintosh computers.

Bus parity signals

The system parity signal (/SP) transmits parity information between NuBus cards that implement NuBus parity checking. Future Apple products may employ this feature, but current Macintosh computers do not provide bus parity checking, so this line is pulled high.

The system parity valid signal (/SPV) indicates that the /SP bit is being used. Cards that do not generate bus parity never drive /SPV active, and cards that do not check parity ignore /SP and /SPV. Future Apple products may employ this feature, but in current versions of the Macintosh computers, this line is pulled high.

◆ *Note:* The Macintosh IIci and the Macintosh IIfx have an optional feature that allows RAM parity checking when 9-bit SIMMs are installed; however, this capability is unrelated to NuBus parity checking.

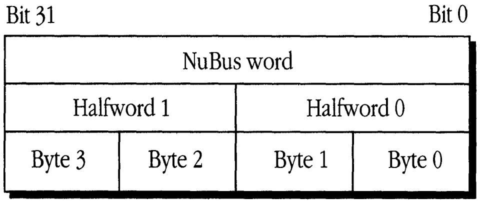
Cache-coherency signals

NuBus '90 defines new signals /CM0, /CM1, /CM2, and /CBUSY to support a cache-coherency protocol. Pins on the Macintosh Quadra–family NuBus connectors are assigned to those signals, but the cache-coherency protocol defined in the NuBus '90 specification is not implemented.

Data-transfer specifications

The NuBus supports reads and writes of several different data sizes. Although optimized for transactions of words and blocks of words, the NuBus also supports byte and NuBus halfword transactions, as shown in Figure 3-1. The base unit of addressability is a NuBus word; /AD31–/AD2 specify the appropriate word. The two least significant address bits (/AD1–/AD0), along with the /TM2–/TM0 signals, specify the transfer mode; that mode determines which part of the addressed word is to be transferred, as shown in Table 3-1. The /TM2 signal is defined in the NuBus '90 specification and is only used for encoding the transfer mode on the Macintosh Quadra 700 and the Macintosh Quadra 900. For all other Macintosh computers with NuBus expansion interfaces, this signal is not defined and is not relevant to specifying the transfer mode.

■ **Figure 3-1** Words, halfwords, and bytes



All NuBus data transfers are unjustified. That is, a byte of data is conveyed on the same byte lane regardless of the transfer mode used to access it. Similarly, a halfword is conveyed on the same halfword lane regardless of the transfer mode used to access it. Therefore, bytes with address 0 modulo 4 are always carried by /AD0 through /AD7, bytes 1 modulo 4 by /AD8 through /AD15, bytes 2 modulo 4 by /AD16 through /AD23, and bytes 3 modulo 4 by /AD24 through /AD31. This unjustified data path approach allows straightforward connection of 8-bit, 16-bit, and 32-bit devices.

Single data cycle transactions

The simplest transactions on the NuBus convey one data item and consist of a start cycle and a subsequent acknowledge cycle. These transactions are either reads or writes of bytes, halfwords, or words.

All transactions are initiated by a bus master, which drives /START active while driving the /TMx, /AD0, and /AD1 signals to define the cycle type. The remaining /ADx signals are also driven to convey the address. The transaction is completed when the responding slave drives /ACK active while driving status information on the /TMx lines. For write transactions, the master must switch the /ADx lines from address to data information in the second clock period and hold that data until acknowledged. In read cycles, the slave drives the data simultaneously with the acknowledge cycle in the last period.

The following abbreviations are used in the timing diagrams and step sequences in this section:

R	Rising (driving) edge of /CLK
F	Falling (sampling) edge of /CLK

Read transactions

Figure 3-2 shows the timing for read bus transactions other than block transfers. Block transfers are implemented only in the Macintosh Quadra family of computers and are not available in other Macintosh computers. Read operations with data widths of 8, 16, and 32 bits are selected by the transfer mode signals (/TMx) and the two low-order address signals (/AD1 and /AD0), as shown in Table 3-1. The slave must put the requested data item on either 8, 16, or all 32 of the /AD31 through /AD0 signals. Any bits other than the requested data may be driven either high or low by the slave; they must be determinate.

Once the bus master has acquired the bus, a read bus transaction involves the following steps:

R(1)[†] The bus master drives /START low, drives /ACK high, and drives the /ADx and /TMx lines with the appropriate values to initiate the transfer.

F(1)[‡] The bus slave samples the /ADx and /TMx lines.

R(2) The bus master releases the /ADx, /TMx, and /START lines and waits for /ACK.

R(*n*)[§] The bus slave drives the requested data onto the /ADx lines, drives the appropriate status code on /TM0 and /TM1, and drives /ACK low.

F(*n*) The bus master samples the /ADx and /TMx lines to receive the data and note any error condition.

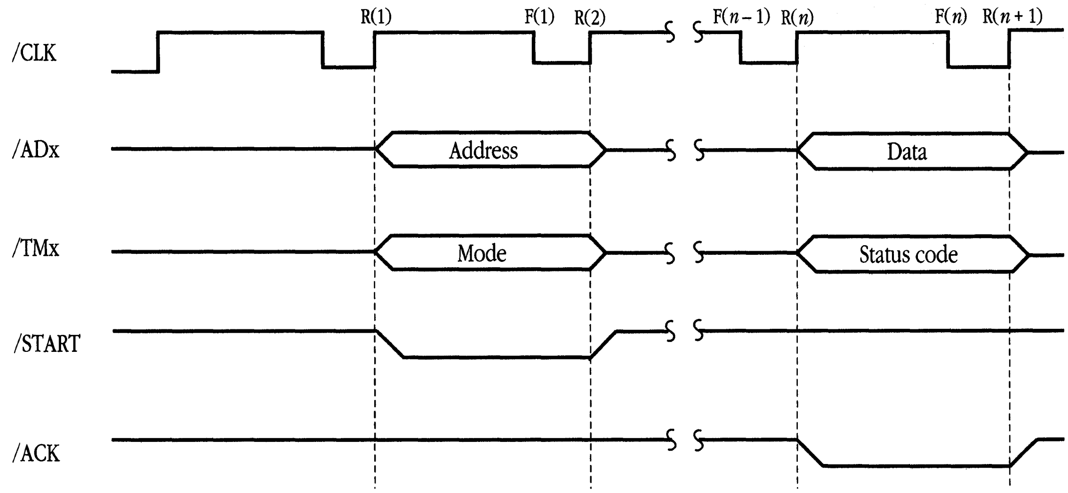
R(*n* + 1) The bus slave releases the /ADx, /ACK, and /TMx lines. This may be the R(1) of the next transaction.

[†] R is the rising edge of /CLK.

[‡] F is the falling edge of /CLK.

[§] $2 \le n < 256$, the system-defined time-out period.

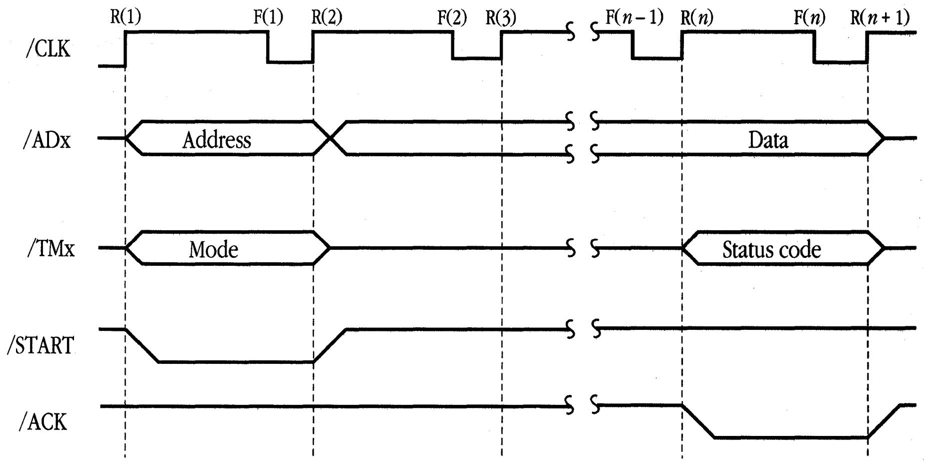
■ **Figure 3-2** Timing of NuBus read transaction



Write transactions

Figure 3-3 shows the timing for write operations other than block transfers. Block transfers are implemented only in the Macintosh Quadra family of computers and are not supported in other Macintosh computers. Write operations with data widths of 8, 16, and 32 bits are selected by the transfer mode signals (/TMx) and the two low-order address bits (/AD1 and /AD0).

- **Figure 3-3** Timing of NuBus write transaction



The bus master has the responsibility for aligning data onto the appropriate /ADx lines for halfword and byte writes. For example, a write of byte 3 requires that the data be placed on /AD24 through /AD31; all other /ADx lines are not defined and are driven to either a high or low state.

Once the bus master has acquired the bus, a write bus transaction involves the following steps:

R(1)[†] The bus master asserts /START and the appropriate /ADx and /TMx lines to initiate the transfer.

F(1)[‡] The bus slave samples the /ADx and /TMx lines.

R(2) The bus master places the data to be written onto the /ADx lines, releases the /START and /TMx lines, and waits for /ACK.

F(2) – F(n)[§] The bus slave samples the /ADx lines to capture the data. The data may be sampled before or during the assertion of /ACK.

R(n) The bus slave asserts /ACK and places the appropriate status code on /TM0 and /TM1 when the data is accepted.

F(n) The bus master samples /ACK and /TMx to determine the end of the transaction.

R(n + 1) The bus master releases the /ADx lines while the bus slave releases the /ACK and /TMx lines.

[†] R is the rising edge of /CLK.
[‡] F is the falling edge of /CLK.
[§] $2 \le n < 256$, the system-defined time-out period.

Acknowledge cycles

During acknowledge cycles the addressed slave drives the /TMx lines while it drives /ACK. The /TMx lines provide status information to the current bus master, as shown in Table 3-2.

■ **Table 3-2** Transfer status coding

/TM1	/TM0	Type of acknowledge
L	L	Bus transfer complete
L	H	Error
H	L	Bus time-out error
H	H	Try again later

Bus transfer complete: The bus transfer complete response indicates the normal valid completion of a bus transaction.

Error: During a read or write operation, certain error conditions may occur. The transaction terminates in a normal manner, and the bus master has the responsibility for handling the error condition reported.

Bus time-out error: If an unimplemented address location is accessed, or for any other reason a slave does not respond to a start cycle address, the attempted transaction is acknowledged with a bus time-out error response. This time-out response indicates that the system-defined time-out period has elapsed while the bus is busy (that is, the bus is between start and acknowledge cycles) and no data transfer acknowledge has occurred. Bus time-out support logic on the Macintosh main logic board enforces a period of 256 clock periods, or 25.6 μs, and assumes the role of the nonresponding slave; it generates an acknowledge cycle with a bus time-out error code.

Try again later: This response status code indicates that a slave is unable to respond at this time to a data-transfer request from a bus master. The master should retry the transaction; slaves should be designed so that a large number of retries are not required.

A Macintosh computer generates a processor bus error exception (/TEA signal on 68040 machines, /BERR signal on others) if its microprocessor attempts a NuBus access that is terminated with an error, a bus time-out, or a try-again-later response.

Attention cycles

An attention cycle is defined as a bus cycle during which both /START and /ACK are asserted. During an attention cycle, the /TMx lines have a different function. The available codings are shown in Table 3-3.

■ **Table 3-3** Attention cycle coding

/TM1	/TM0	Type of attention cycle
L	L	Attention-null
L	H	Reserved
H	L	Attention-resource-lock
H	H	Attention-cache

Attention cycles can be used to reinitiate bus arbitration (attention-null), to signal a resource lock (attention-resource-lock), or both. Refer to Chapter 4, "NuBus Arbitration," for a detailed explanation of bus arbitration and resource locking.

Attention-null: The attention-null cycle has two uses:

- to reinitiate arbitration after the bus has been requested and won, but the new bus owner decides not to transfer data (in this case, the new bus owner must generate an attention-null cycle)
- to indicate the end of a data transfer using a locked resource

During an attention-null cycle, the /ADx lines are ignored by all bus cards, and no data may be transferred.

Attention-resource-lock: An attention-resource-lock cycle occurs at the beginning of a sequence of locked transactions constituting a locked tenure of the current bus master. During this tenure, cards with lockable multiport resources lock them against access by local processors other than the NuBus master. That tenure is terminated by an attention-null cycle. During an attention-resource-lock cycle, the /ADx lines are ignored by all bus cards, and no data may be transferred.

Attention-cache: The attention-cache cycle is used in cache-coherent transactions. The master drives the /ADx and /TMx lines with the desired address, block size, and transfer mode. The attention-cache cycle will reinitiate arbitration, but does not affect the state of any resource lock. *NuBus cache-coherency transactions are not currently supported on any of the Macintosh computers.*

You should follow these implementation rules:

- Masters must drive /ACK high during their start cycle to guarantee that /ACK is in the unasserted state and the start cycle is not interpreted as an attention cycle.
- Masters must ensure that the first /ACK terminates a transaction. An attention cycle immediately following the acknowledge cycle *must not* latch data.
- Slaves must qualify /START with the logical complement of /ACK to decode a start cycle. Otherwise, an attention cycle could be misinterpreted as a start cycle.

Interrupt operations

Three possible ways to handle NuBus interrupts are available, but only one way is used by the Macintosh computers with the NuBus interface.

By write transaction

Interrupts on the NuBus can be implemented as write transactions. *Interrupts are not done this way on Macintosh computers.* Interrupt operations require no unique signals or protocols. Any card on the NuBus that is capable of becoming bus master can interrupt a processor card by performing a write operation into an area of memory that is monitored by that processor. Any address range on the processor card can be defined as its interrupt space. This allows interrupts to be posted to individual processors and allows interrupt priority to be software specified by memory mapping the priority level.

By slots sharing a single NuBus /NMRQ line

The individual slot /NMRQ (nonmaster request) signals may drive a single NuBus line (/NMRQ), in which case, the system processor will have available only the wired-OR result of all of the slot /NMRQ signals. In this case, the software must poll the slots capable of generating the bus /NMRQ signal to determine the source or sources of the interrupt. *Interrupts are not done this way on Macintosh computers.*

By a dedicated /NMRQ line from each slot

Macintosh computers with NuBus use a separate (non-NuBus) /NMRQ line from each slot to support interrupts (see Figures 1-1 through 1-7). Each card slot has a unique /NMRQ line driving an OR gate whose output is a real hardware interrupt signal to the microprocessor (through VIA2, or equivalent). In addition, each of the /NMRQ lines can be independently polled by the processor, to allow the software to communicate with the appropriate handlers for each of the cards asserting /NMRQ. NuBus expansion cards must keep the /NMRQ signal asserted until the interrupt service routine gets called. The interrupt service routine must clear the interrupt.

1X block data transfers

Single-rate block transfers, or 1X block transfers, have not been implemented in most Macintosh computers with NuBus, and have only recently been provided in the Macintosh Quadra 700 and the Macintosh Quadra 900. Keep in mind that the following discussion applies only to the NuBus implementation in the Macintosh Quadra family–computers.

A 1X block transfer is a read or write transaction in which multiple data values are transferred. A 1X block transfer consists of a start cycle, multiple data cycles to or from sequential address locations, and an acknowledge cycle. The number of data cycles is controlled by the master and communicated during the start cycle. Allowed lengths of 1X block transfers are 2, 4, 8, and 16 words. (Only 32-bit NuBus word transfers are supported in block-transfer mode.)

The /TMx and /ADx encoding for 1X block transfers is shown in Table 3-1. The starting address of the block must correspond to the size of the block and is encoded by the /AD2 through /AD5 lines, as shown in Table 3-4.

During a 1X block transfer, each data cycle is acknowledged by the responding slave. The intermediate acknowledges are data cycles where /TM0 is asserted and /TM1 and /ACK are both unasserted. For intermediate acknowledgments, /TM0 has the same significance and timing as the /ACK signal for nonblock transfers. The acknowledgment of the final word transfer is a standard acknowledge cycle. Status codes are shown in Table 3-2.

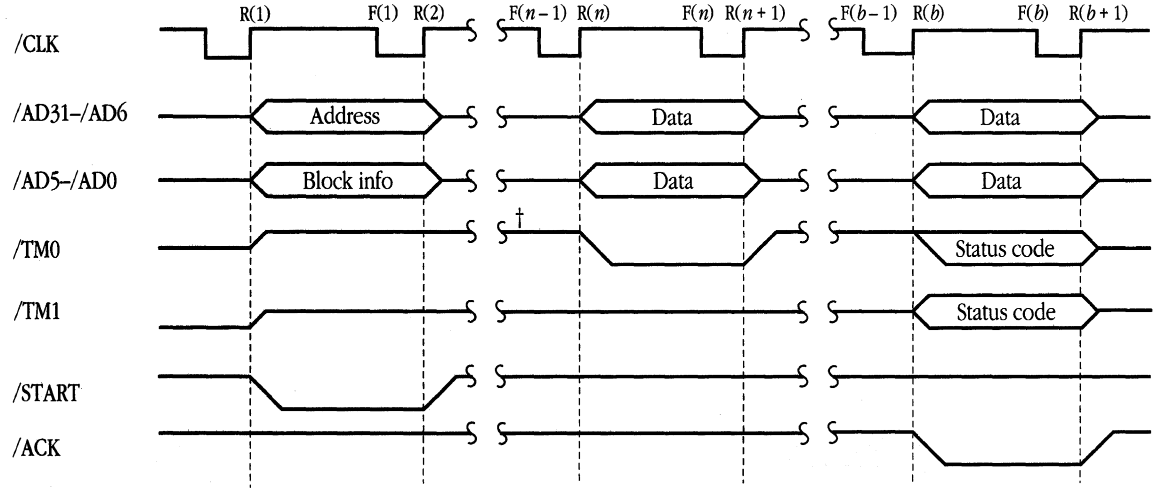
■ **Table 3-4** Block size and starting address coding for 1X block transfers

/AD5	/AD4	/AD3	/AD2	Block size (words)	Block starting address
—	—	—	H	2	(AD31–AD3)000
—	—	H	L	4	(AD31–AD4)0000
—	H	L	L	8	(AD31–AD5)00000
H	L	L	L	16	(AD31–AD6)000000
L	L	L	L	Error	

1X block read

Figure 3-4 shows the timing for a NuBus 1X block read transaction. See Table 3-1 for the /TMx and /ADx encoding that initiates block reads. The /AD5 through /AD2 lines determine the size and starting address of the transaction, as shown in Table 3-4. The responding slave drives data onto the bus, and the initiating bus master accepts the data on each intermediate or final acknowledge. Assertion of /TM0 is used by the responding slave as an intermediate acknowledge, meaning that the next consecutive word of data is ready to be put on the bus.

- **Figure 3-4** Timing of NuBus 1X block read transaction



† The addressed slave is responsible for driving /TM0 to the desired state between R(*n*) and R(*b* + 1).

Once the bus master has acquired the bus, a 1X block read consists of these steps:

R(1)[†] The bus master asserts /START and the appropriate /ADx and /TMx lines to initiate the transfer.

F(1)[‡] The bus slave samples the /ADx and /TMx lines.

R(2) The bus master releases the /ADx, /TMx, and /START lines and waits for an intermediate acknowledge (/TM0 asserted).

R(n)[§] The bus slave places the first word of requested data on the /ADx lines and asserts /TM0.

F(n) The bus master samples the /ADx lines and /TM0 to capture data. The /TM0 signal is asserted and the first word of data is captured.

R(n + 1) If the next consecutive word of data is not ready to be put on the bus, the slave drives /TM0 unasserted until the word is ready.

The previous three steps are repeated for ascending addresses until $B - 1$ words have been transferred, where B is the block size (2, 4, 8, or 16).

R(b)[¶] The bus slave places the final word of requested data onto the /ADx lines, asserts /ACK, and places the appropriate status code on /TM0 and /TM1.

F(b) The bus master samples the /ADx and /TMx lines to receive the data and note any error conditions.

R(b + 1) The bus slave releases the /ADx, /ACK, and /TMx lines.

[†] R is the rising edge of /CLK.

[‡] F is the falling edge of /CLK.

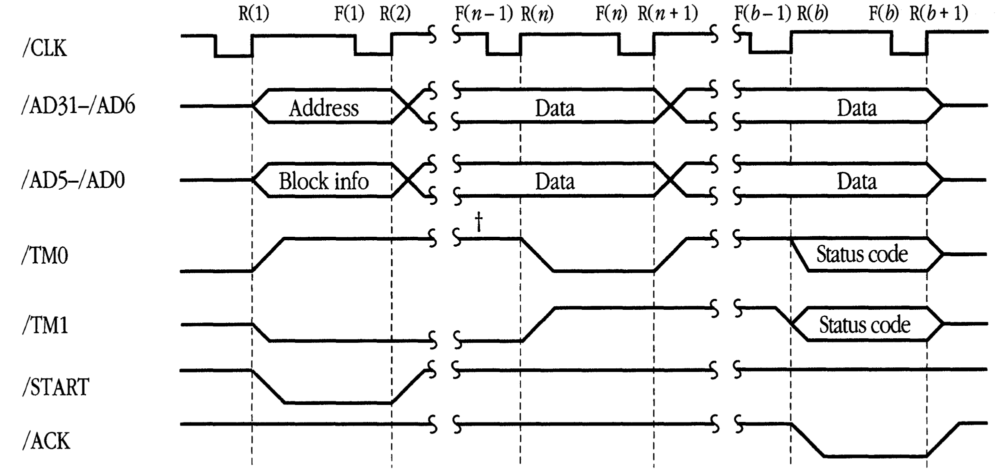
[§] $2 \leq n < 256$, the system-defined time-out period.

[¶] $2 \leq b \leq 256B$.

1X block write

Figure 3-5 is a timing diagram for a NuBus 1X block write operation. Block writes are similar to block reads except the bus master drives the data bus while the slave accepts data. The format for describing block size and starting address is the same as for block read.

■ **Figure 3-5** Timing of NuBus 1X block write transaction



† The addressed slave is responsible for driving /TM0 to the desired state between R(n) and R(b + 1).

Once the bus master has acquired the bus, a 1X block write consists of these steps:

R(1)† The bus master asserts /START and the appropriate /ADx and /TMx lines to initiate the transfer.

F(1)‡ The bus slave samples the /ADx and /TMx lines.

R(2) The bus master places the data to be written onto the /ADx lines, stops driving the /ACK and /TMx lines, drives /START unasserted, and waits for an intermediate acknowledge (/TM0 asserted).

R(*n*)§ The bus slave asserts /TM0 when the first word of data is accepted.

F(*n* – 1) The bus slave samples the /ADx lines to capture the data being written. The data may be sampled before or during the assertion of /TM0.

R(*n* + 1) The bus master places the next consecutive word of data on the bus.

The previous three steps are repeated for ascending addresses until $B - 1$ words have been transferred, where B is the block size.

R(b)[¶] The bus slave asserts /ACK and places the appropriate status code on /TM0 and /TM1 when the final word of data is accepted.

F(b–) The bus slave samples the /ADx lines to capture the data. The b– notation implies the data may be sampled before or during the assertion of /ACK.

R(b + 1) The bus master releases the /ADx lines while the bus slave releases the /ACK and /TMx lines.

[†] R is the rising edge of /CLK.

[‡] F is the falling edge of /CLK.

[§] $2 \le n < 256$, the system-defined time-out period.

[¶] $2 \le b \le 256B$.

1X block transfer errors

Although the length of a 1X block transfer is dictated by the master during the start cycle, a 1X block transfer may be cut short by an error acknowledgment from the slave at any time. The standard status codes shown in Table 3-2 are used.

The speed of a 1X block transfer is controlled by the slave; therefore, a master requesting a 1X block transfer must be capable of transferring data at the speed of the *fastest* slave in the system. This could be one word per NuBus clock cycle (one word per 100 ns). If the master is incapable of transfers at the speed the slave specifies, an *undetectable* overrun (or underrun) occurs.

NuBus specifies that if a slave supports 1X block transfers, it must support all types of data transfer (byte, halfword, and word). In the case of a 1X block transfer request to a slave that cannot support block transfers, that slave should terminate the first transfer with /ACK and a normal status code. This is *not* considered an error condition. The data should be ignored for read or write purposes, but the master shall not assume that the data transfer did not take place.

2X block data transfers

A 2X block transfer, or double-rate transfer, is a read or write transaction that is twice the speed of a 1X block transfer. Double-rate block transfers are a new feature in the NuBus '90 specification. Block transfers have only recently been provided in the Macintosh Quadra 700 and the Macintosh Quadra 900, and although computers in the Macintosh Quadra family allow double-rate block transfers between NuBus cards, they do not support double-rate transfers to or from main memory.

In 2X block transfers, two NuBus words are transferred in a single NuBus cycle, allowing a word to be transferred every 50 ns. The 2X block-transfer protocol also provides several new capabilities not available with the 1X block-transfer protocol:

- longer transfer sizes (up to 256 words)
- autosizing, which allows a slave to prematurely terminate a transfer that is too long
- master flow control (slave flow control is available with both 1X and 2X block transfers)

Signal protocol for 2X block transfers

Two additional signals, /CLK2X and /TM2, have been introduced for 2X block transfers:

- /CLK2X synchronizes the 2X block data transfers. The falling (assertion) edge of /CLK2X is coincident with the rising edge of /CLK. During a 2X block transfer, modules drive new data on the assertion edge of /CLK2X and sample the data on the following assertion edge.
- In conjunction with the block read and block write encoding of the /TMx and /ADx lines at the beginning of a bus transaction, the bus master asserts /TM2 to request that the addressed slave perform a 2X block transfer. Unlike the /TMx lines, the master continues to drive /TM2 during the remainder of the 2X block transfer, using it as a "master flow control" signal for pairs of NuBus words.

The transfer mode signals, /TM0 and /TM1, also participate in a 2X block transfer. After the start cycle of a 2X block transfer, the slave asserts /TM1 to indicate that it can perform a 2X block transfer. The slave drives /TM1 unasserted during all intermediate acknowledges (when /TM0 is asserted) if it is not capable of performing 2X block transfers. At the end of the 2X block transfer, the addressed slave drives /TM1 unasserted to indicate that the next pair of words will be the last. This allows the slave to autosize, or prematurely terminate, a 2X block transfer if it cannot transfer the requested amount of data in a single transaction.

The start cycle and first word of a 2X block transfer use 1X block-transfer timing. If the slave indicates that it can perform a 2X transfer by asserting /TM1, subsequent words are transferred as pairs, and the last word is transferred during the acknowledge cycle. The less stringent timing for the first and last words provides additional time to establish the direction of the data transfer.

The number of words transferred is controlled by the master and communicated to the slave during the start cycle. The allowed lengths of 2X block transfers are 4, 8, 16, 32, 64, 128, and 256 words. Only word transfers are provided in 2X and 1X block mode. Note that the total duration of the transfer shall not exceed 256 /CLK cycles (or 25.6 µs).

The size of the block to be transferred and its starting address are determined by an encoding of the /ADx lines, as defined in Table 3-5. The encoding is identical to the one used for 1X block transfers except that two word transfers are not permitted and transfers of 32, 64, 128, and 256 words are permitted.

■ **Table 3-5** Block size and starting address coding for 2X block transfers

		/ADx lines						Block size,	Block
09	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**words**	**starting address**
—	—	—	—	—	—	H	L	4	(AD31–AD4)0000
—	—	—	—	—	H	L	L	8	(AD31–AD5)00000
—	—	—	—	H	L	L	L	16	(AD31–AD6)000000
—	—	—	H	L	L	L	L	32	(AD31–AD7)0000000
—	—	H	L	L	L	L	L	64	(AD31–AD8)00000000
—	H	L	L	L	L	L	L	128	(AD31–AD9)000000000
H	L	L	L	L	L	L	L	256	(AD31–AD10)0000000000

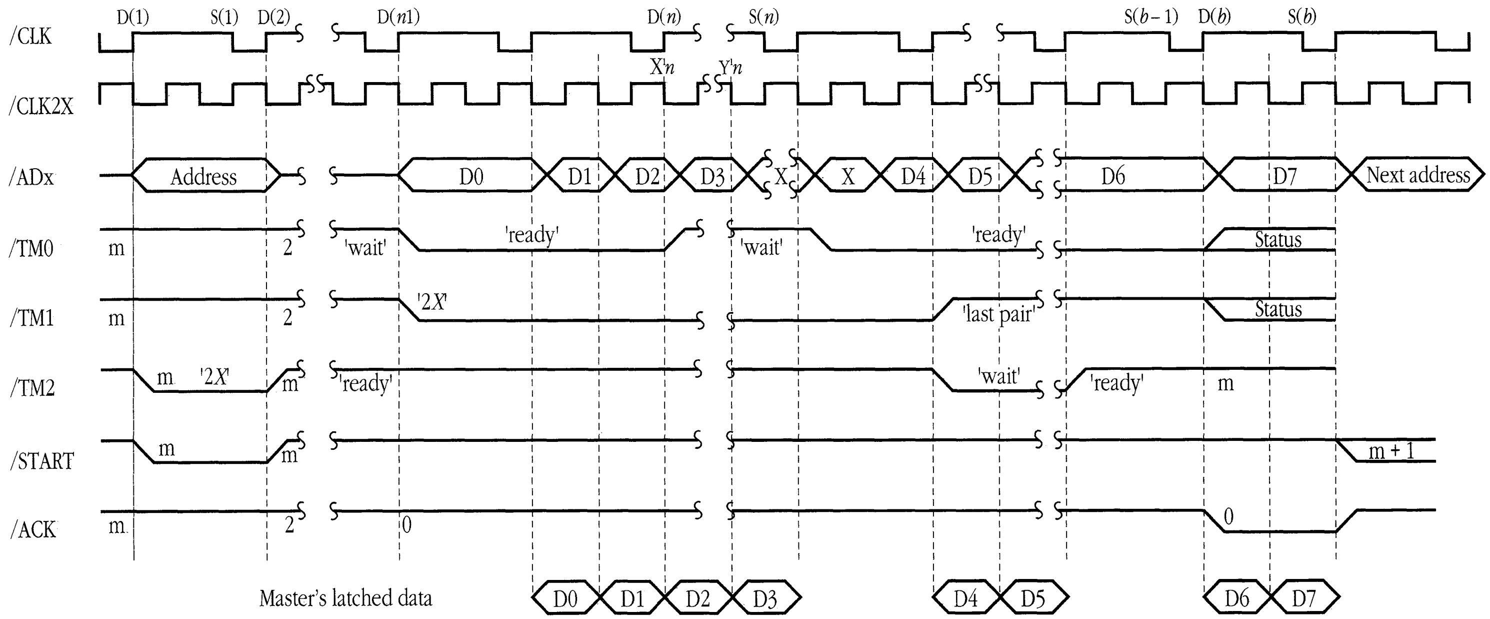
2X block-transfer flow control

Master and slave flow control are provided during 2X block transfers, whereas only slave flow control is provided during 1X block transfers. Allowing both the master and the slave to control the transfer of data makes larger burst transfers possible without the need for large buffers or frequent arbitration for NuBus and local board resources.

During 2X block transactions, both the master and the slave control the rate of transfer using the /TM2 and /TM0 lines, respectively. During a 2X block read transaction, the slave uses the /TM0 signal to indicate when it is ready to send the data, and the master uses the /TM2 signal to indicate when it can receive the data. During a 2X block write transaction, the bus master uses the /TM2 signal to indicate when it can send the data, and the slave uses the /TM0 signal to indicate when it can receive the data.

2X block read transfer

Figure 3-6 shows the timing diagram for a NuBus 2X block read transaction. See Table 3-1 for the /TMx, /AD0, and /AD1 encoding that initiates 2X block reads. The /ADx lines determine the starting address of the transaction, as shown in Table 3-5.



■ **Figure 3-6** Timing of NuBus 2X block read transaction

Once the bus master has acquired the bus, a 2X block read consists of these steps:

D(1) The bus master asserts /START and the appropriate /ADx and /TMx lines to initiate the transfer.

S(1) The bus slave samples the /ADx and /TMx lines. Bus modules capable of supporting 2X block transfers sample the /TM2 line.

D(2) The bus master stops driving the /ADx, TMx, and /ACK lines. The master drives /TM2 unasserted, indicating that it is ready to receive the first pair of words. The master drives /START unasserted and waits for an intermediate acknowledgment (/TM0 asserted).

D(*n*1) The bus slave asserts /TM0 when the first word of data is accepted. If the slave is capable of supporting 2X block transfers, it asserts /TM1 and drives /ACK unasserted.

If the bus master issues a 2X block-transfer request of 16 words or less to a slave that can only support 1X block transfers, the addressed slave should respond by driving /TM1 and /ACK unasserted, with /TM1 unasserted during all intermediate acknowledges (when /TM0 is asserted), and the block transfer will be completed using the 1X block transfer.

If the bus master issues a 2X block-transfer request of 32 NuBus words or more to a slave that cannot support 2X block transfers, the slave issues an immediate acknowledgment cycle with a bus-transfer complete status code.

D(*n*) The bus master drives /TM2 unasserted to indicate that it can accept the next pair of words, or asserted to indicate that it cannot accept the next pair of words.

The slave drives /TM0 asserted to indicate that it can send the next pair of words, or unasserted to indicate that it cannot send the next pair of words.

The slave continues to assert /TM1 until it is ready to transfer the last pair of NuBus words. The slave drives /TM1 unasserted (coincident with the assertion of /TM0) if the next pair of words is the last pair to be transferred.

y(*n*) If the slave asserted /TM0 during this cycle, the slave drives the first word of the next pair on /ADx lines after an output hold delay.

S(*n*) The bus master samples /TM0 and the slave samples /TM2 to determine if the next pair of NuBus words should be transferred.

x(*n* + 1) If /TM0 is asserted and /TM2 is unasserted (data is transferred), the master accepts the first word of the pair. After an output hold delay, the slave drives the second word of the pair on /ADx lines.

y(*n* + 1) If /TM0 is asserted and /TM2 is unasserted, the master accepts the second word of the pair. The slave drives the first word of the next pair on /ADx lines after an output hold delay.

D(*n* + 1) If the slave is not ready to send the next pair of NuBus words, the slave drives /TM0 unasserted until it is ready to send the next pair of words.

The previous steps starting at D(*n*) are repeated for ascending addresses until all but the final pair of NuBus words have been transferred.

S(*b* – 1) The master, driving /TM2 unasserted and responding to the assertion of /TM0 and /TM1 and /ACK unasserted, accepts the last pair of words as shown here:

- x(*b*) The master accepts the first word of the final pair. After an output hold delay, the slave drives the final NuBus word on the /ADx lines.
- y(*b*) The master accepts the final NuBus word. The slave continues to drive the last word of data on the /ADx lines.

D(*b*) In addition to transferring the last pair of words, the slave drives the appropriate transfer response on /TM0 and /TM1 and drives /ACK asserted.

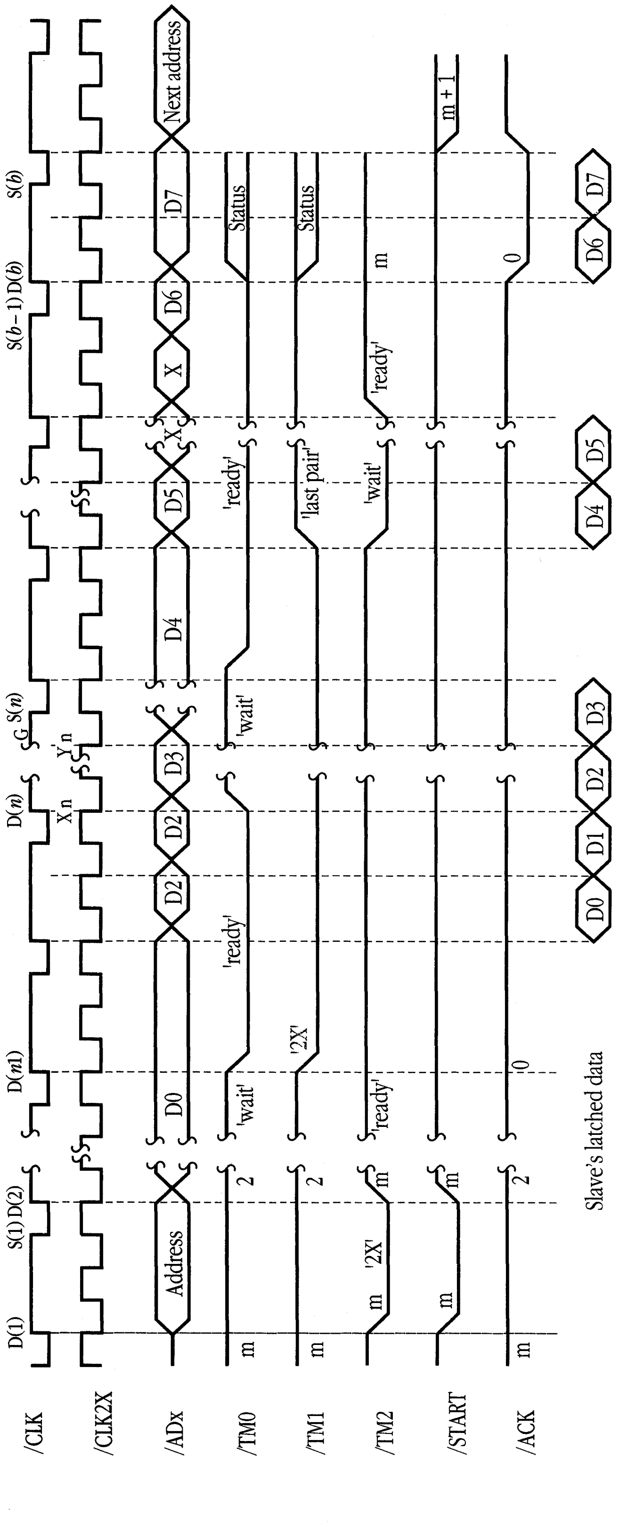
S(*b*) The master samples /TM1 and /TM0 to note any error conditions.

D(*b* + 1) The slave stops driving the /ADx, /TM1, /TM0, and /ACK lines, and the master stops driving the /TM2 line. The bus owner drives /ACK to a determinate state. This may be the D(1) of the next transaction.

2X block write transfer

Figure 3-7 shows the timing diagram for a NuBus 2X block write transaction. See Table 3-1 for the /TMx, /AD1, and /AD0 encoding that initiates 2X block writes. Double-rate block writes are similar to block reads except the bus master drives the data bus while the slave accepts the data. The format for describing block size and starting address is the same for a 2X block read and is shown in Table 3-5.

■ **Figure 3-7** Timing of NuBus 2X block write transaction



Once the bus master has acquired the bus, a 2X block write consists of these steps:

D(1) The bus master asserts /START and the appropriate /ADx and /TMx lines to initiate the transfer. The master asserts /TM2 to request a 2X block transfer.

S(1) The bus modules sample the /ADx and /TMx lines. Bus modules capable of supporting 2X block transfers sample the /TM2 line.

D(2) The bus master drives the first NuBus word to be written on /ADx lines and stops driving the /TM0, /TM1, and /ACK lines. The master drives /TM2 unasserted, indicating that it is ready to transmit the first pair of data words. The master drives /START unasserted and waits for an intermediate acknowledgment on /TM0.

D(*n*1) If the slave is capable of supporting 2X block transfers, it asserts /TM1 and drives /ACK unasserted.

If the bus master issues a 2X block-transfer request of 16 words or less to a slave that can only support 1X block transfers, the addressed slave should respond by driving /TM1 and /ACK unasserted, with /TM1 unasserted during all intermediate acknowledges (when /TM0 is asserted), and the block transfer will be completed using the 1X block transfer.

If the bus master issues a 2X block-transfer request of 32 words or more to a slave that cannot support 2X block transfers, the slave issues an immediate acknowledgment cycle with a bus-transfer complete status code.

D(*n*) The slave drives /TM0 asserted to indicate that it can accept the next pair of NuBus words, or unasserted to indicate that it cannot accept the next pair of words.

The slave continues to assert /TM1 until it is ready to transfer the last pair of words. The slave drives /TM1 unasserted (coincident with the assertion of /TM0) if the next pair of words is the last to be transferred.

The bus master drives /TM2 unasserted to indicate that it can send the next pair of NuBus words, or asserted to indicate that it cannot send the next pair of words.

y(*n*) If the master drives /TM2 unasserted during this cycle, the master drives the first word of the next pair on the /ADx lines after an output hold delay.

S(*n*) The bus master samples /TM0, and the slave samples /TM2 to determine if the next pair of words should be transferred.

x(*n* + 1) If /TM0 is asserted and /TM2 is unasserted, the master drives the second word of the pair on the /ADx lines.

y(*n* + 1) If /TM0 is asserted and /TM2 is unasserted, the slave accepts the second word of the pair. The master drives the first word of the next pair on the /ADx lines after an output hold delay.

D(*n* + 1) If the slave is not ready to accept the next pair of words, the slave drives /TM0 unasserted until a bus cycle in which it is ready to send the next pair of words. If the master is not ready to send the next pair of words, the master drives /TM2 asserted until a bus cycle in which it is ready to send the next pair of words.

The previous steps, starting at D(*n*), are repeated for ascending addresses until all but the final pair of words has been transferred.

S(*b* – 1) The master, driving /TM2 unasserted and responding to the assertion of /TM0 and /TM1 and /ACK unasserted, sends the last pair of NuBus words as shown here:

> x(*b*) The slave accepts the first word of the final pair. After an output hold delay, the master drives the final word on the /ADx lines.
>
> y(*b*) The slave accepts the final word of data. The master continues to drive the last word of data on the /ADx lines.

D(*b*) In addition to accepting the last pair of NuBus words, the slave drives the appropriate transfer response on /TM0 and /TM1 and drives /ACK asserted.

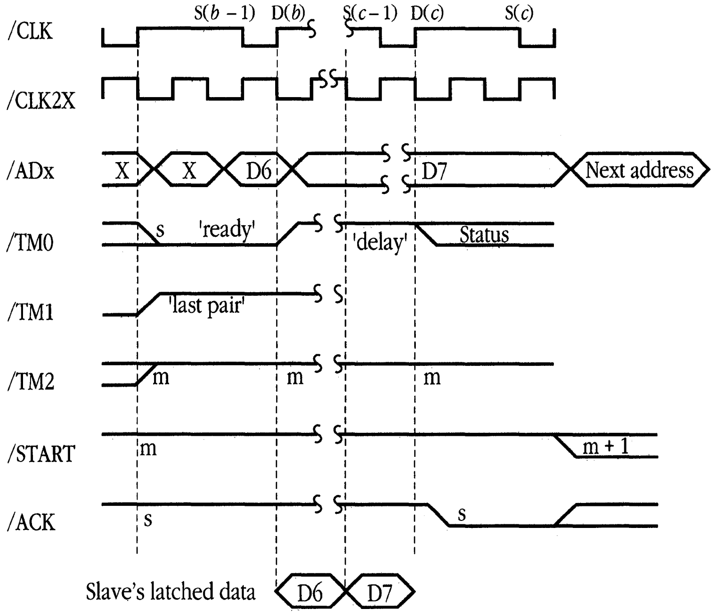
S(*b*) The master samples /TM1 and /TM0 to note any error conditions.

D(*b* + 1) The master stops driving the /ADx and /TM2 lines, and the slave stops driving the /TM1, /TM0, and /ACK lines. The bus owner drives /ACK to a determinate state. This may be the D(1) of the next transaction.

2X block write transfer with delayed status indication

To allow the slave to report parity errors on 2X block write transfers, the slave can defer the acknowledgment cycle until the last word of the transfer has been received and its parity has been evaluated. The capability is optional for 2X slaves, but all 2X bus masters must be able to accept a deferred acknowledgment after a 2X block write transfer. Deferred acknowledgments are not permitted after a 2X block read transfer. Figure 3-8 shows the timing diagram for a 2X block write with delayed indication.

■ **Figure 3-8** Timing of NuBus 2X block write with delayed status indication



Once the bus master has acquired the bus, a 2X block write with delayed status indication consists of the following steps:

S(*b* – 1) The master, driving /TM2 unasserted and responding to the assertion of /TM0 and /TM1, and /ACK unasserted, sends the last pair of data words as shown here:

- x(*b*) The slave accepts the first word of the final pair. After an output hold delay, the master drives the final word on the /ADx lines.
- y(*b*) The slave accepts the final word of data. The master continues to drive the last word of data on the /ADx lines.

D(*b*) In addition to accepting the last pair of data words, the slave drives /TM0 and /TM1, and /ACK unasserted.

S(*b*) – S(*c*) The master, driving /TM2 unasserted, waits until the slave asserts /ACK.

D(*c*) The slave drives the appropriate transaction response status on the /TM1 and /TM0 lines and asserts /ACK.

S(*c*) The master, driving /TM2 unasserted and responding to the assertion of /ACK by the slave, samples /TM1 and /TM0 to note any error conditions.

D(*c* + 1) The master stops driving the /ADx and /TM2 lines, and the slave stops driving the /TM1, /TM0, and /ACK lines. The bus owner drives /ACK to a determinate state. This may be the D(1) of the next transaction.

Cache coherency

Cache coherency is an optional NuBus '90 protocol used to determine the ownership of data between multiple processors with private caches and to maintain coherence, or agreement, between data shared by the processors and shared memory. New signals /CM0, /CM1, /CM2, and /CBUSY have been defined to support a NuBus cache-coherency protocol.

△ **Important** *NuBus cache-coherent transactions are not currently implemented in any members of the Macintosh family,* but they may be implemented in future Apple products. Apple has chosen to implement cache coherency differently. The following discussion, although not pertinent to the Macintosh computer, is provided for completeness in describing the NuBus. An overview of cache coherency has been provided. For more details, please refer to the NuBus '90 specification. △

Shared memory may be physically located on a processor module but can be accessed by other processors via NuBus. In other implementations, the shared memory may be physically located on an external memory module that is accessed only via NuBus.

The use of a **cache memory** for each processor can greatly reduce bus traffic since the majority of memory references can be serviced by the cache. Bus traffic can be further reduced by using a **copyback cache,** where the data in the cache is written to shared memory only if the cache is full, or during a cache flush. Cache memories typically transfer blocks of data, referred to as a **cache line.**

Since multiple caches can have simultaneous copies of data for a given memory location, a cache-coherency protocol is required to ensure that all copies remain consistent.

In this section, you can find information about the cache line states, cache snooping, transactions in the cache-coherency protocol, cache-coherent states, and arbitration by cache-coherent modules.

Cache line states and sizes

A cache line can be in one of four states defined below.

- A cache line is *modified* if the data may be different from memory and there is only one cached copy of the line in the system.
- A cache line is *exclusive* if the data is known to be the same as memory and there is only one cached copy of the line in the system.
- A cache line is *shared* if the data is known to be the same as memory and another cached copy of the line may exist somewhere in the system.
- A cache line is *invalid* if there is not an up-to-date copy of the line in the module's cache.

A cache line can be 4, 8, 16, 32, or 64 bytes long, and caches with different cache line sizes can be used at the same time. The addresses of cache lines are aligned to their cache size.

Read and write miss

When a processor issues a read and the requested location is exclusive, shared, or modified, the operation is said to be a **read hit.** The processor can use the data in the cache and no bus transaction is necessary. If the location is invalid, the operation is said to be a **read miss.**

When a processor issues a write and the requested location is modified or exclusive, the operation is said to be a **write hit.** The processor can modify the data in the cache and no bus transaction is necessary. If the location is shared or invalid, the operation is said to be a **write miss,** even though the location may be valid.

Snooping

A snooping module is one that snoops, or monitors, cache-coherent transactions between a master and a slave. This can include cache tag comparison, reporting cache line status to the master, and, if necessary, writing modified data to shared memory. Snooping modules monitor the /CMx signal lines to detect the cache-coherence start cycle and monitor the address and block size specified by the /ADx and /TMx lines. The response of the snooping module is based on the initial state of the cache line (modified, exclusive, shared, or invalid) and the transaction mode indicated by /CM1 and /CM2. The response of the snooping module does not depend on the type of data transaction. In some cases, the snooping module may be required to write back a modified cache line to update memory. After a snooping module examines its cache line tags and performs a write-back transaction (if necessary), it updates the cache line tag to its final state.

Cache-coherency transactions

Cache-coherent modules use the existing bus arbitration protocol to request bus ownership and an additional protocol and signal line to ensure fair access to the bus by all cache-coherent modules. Cache-coherent transactions are initiated by a master, are observed by snooping modules, and are completed by the addressed slave. During the cache-coherent start cycle, the /CM1 and /CM0 signal lines specify the cache-update transaction, and the /TMx signal lines specify the data transaction. Table 3-6 summarizes the cache-coherent transactions.

■ **Table 3-6** Cache-coherent transactions

/CM1	/CM0	/TM1	/TM0	Transaction
L	L	H	L	`ReadExclusive`
L	L	H	H	`AttentionExclusive`
L	L	L	—	`WriteExclusive`
L	H	H	L	`ReadShared`
L	H	H	H	`AttentionShared`
L	H	L	—	(not used)
H	L	H	L	`ReadInvalidate`
H	L	H	H	`AttentionInvalidate`
H	L	L	—	`WriteInvalidate`
H	H	H	L	`ReadNosnoop`
H	H	H	H	(not used)
H	H	L	—	`WriteNosnoop`

All cache-coherent transactions begin with a cache-coherent start cycle that is coincident with the /START cycle of a data-transfer or attention-cache cycle. During the cache-coherent start cycle, the /CM0 and /CM1 signal lines specify the cache-update transaction (exclusive, shared, invalidate, or nosnoop), and the /TMx lines specify the data transaction (read, write, or attention).

Read and write transactions may use single word transfers or 1X and 2X block transfers, depending on the capabilities of the master and the slave. The attention transactions are address-only transactions in which the master asserts both /ACK and /START during the same cycle. They can be used by processors accessing physically local but globally coherent memory, for flushing or purging data, or in other situations where an initial read or write data transaction is not required.

The initial read, write, or attention transaction may be followed by one or more data transactions by snooping modules that write back modified data to shared memory before the original master updates its cache with the updated data in memory.

The `ReadExclusive`, `AttentionExclusive`, and `WriteExclusive` transactions guarantee that at the end of the transaction the master has the only copy of the data. The `ReadShared` and `AttentionShared` transactions guarantee that at the end of the transaction the master's copy of the data is the same as memory. The `ReadInvalidate`, `AttentionInvalidate`, and `WriteInvalidate` transactions guarantee that at the end of the transaction no other cache in the system has a copy of the data. Finally, the `ReadNosnoop` and `WriteNosnoop` transactions inhibit snooping by all modules that could potentially snoop the transaction.

The following sections give a brief description of each transaction. For more information and for timing diagrams for each transaction, please refer to the NuBus '90 draft specification.

ReadShared

A `ReadShared` transaction is used when a processor cache read miss occurs and data must be read from an external shared memory module. At the end of the transaction, the processor's cache and shared memory will contain valid copies of the data, and possibly one or more external caches may contain valid copies of the data.

ReadExclusive

The `ReadExclusive` transaction can be used to service a processor cache write miss by ensuring that the cache line is exclusive before it is modified. The shared memory is located on an external memory module.

ReadInvalidate

The ReadInvalidate transaction may be used by a master to read shared memory and invalidate any cache lines that reference data. The use of the ReadInvalidate transaction assumes that the processor previously copied back to memory the set of modified cache lines that correspond to areas of memory that will be read.

ReadNosnoop

The ReadNosnoop transaction may be used by a master, such as a DMA I/O controller, to read shared memory with snooping inhibited. The used of the ReadNosnoop transaction assumes that the processor has previously copied back to memory the set of modified cache lines that correspond to areas of memory that will be read using the ReadNosnoop transaction. Alternatively, the ReadNosnoop transaction can be used to read data from areas of memory that use a write-through cache-coherence policy, where the cache controller always writes to both the cache and main memory. Since snooping is inhibited, the state of any cache lines is unaffected.

The ReadNosnoop transaction can also be used to access read-only data from memory, such as instruction code and data that cannot be modified. After the transaction is performed, the master marks its cache line shared.

WriteExclusive

The WriteExclusive transaction can be used for writing partial cache lines to external shared memory. The master first writes the data, typically a single word, to shared memory. If none of the snooping modules have modified data, the WriteExclusive transaction is complete. Otherwise, the master waits until all snooping masters have updated shared memory with their modified data, and then retries the original write transaction. At the end of the transaction, the processor's cache line will be in the exclusive state. An alternative to the WriteExclusive transaction is a ReadExclusive transaction followed by a WriteNosnoop (copyback) transaction.

WriteInvalidate

The WriteInvalidate transaction may be used by a master, such as a DMA I/O controller, to invalidate all copies of the cache line and write a full cache line to memory. The previous contents of the cache line are not needed since the entire cache line is replaced by new data. At the successful completion of the transaction, no cache in the system contains a valid copy of the cache line.

WriteNosnoop

The WriteNosnoop transaction may be used by a master, such as a DMA I/O controller, to write to shared memory with snooping inhibited. The use of the WriteNosnoop transaction assumes that the processor has previously invalidated the set of cache lines that correspond to areas of memory that will be written using the WriteNosnoop transaction. Since snooping is inhibited, the state of any cache lines is unaffected.

The WriteNosnoop transaction can also be used to copy back a modified cache line to external shared memory to free up space in the cache. Snooping is not required since the cache has an exclusive copy of the data. After the copyback operation is performed, the master marks its cache line either invalid or exclusive.

AttentionShared

The AttentionShared transaction is similar to ReadShared except that it is used by masters accessing physically local but globally coherent memory. The AttentionShared transaction is used when a processor cache read miss occurs and data must be read from shared memory physically located on the processor module. At the end of the transaction, the processor's cache and shared memory will contain valid copies of the data, and possibly one or more external caches will contain valid copies of the data.

AttentionExclusive

The AttentionExclusive transaction is similar to ReadExclusive except that it is used by masters accessing physically local but globally coherent memory. The AttentionExclusive transaction can be used to service a processor cache write miss by ensuring the cache line is exclusive before it is modified. The shared memory is physically located on the processor module.

AttentionInvalidate

The AttentionInvalidate transaction is used by a caching master to service a write hit to a shared cache line by invalidating all other caches and changing its cache line status from shared to modified. All other caches are invalid after the completion of the AttentionInvalidate transaction.

Non-cache-coherent transactions to caching modules

It is recommended that cache-coherent modules support accesses by noncaching bus masters. This includes DMA I/O controllers and earlier bus master designs that can only issue non-cache-coherent read and write transactions.

It is recommended that memory locations accessed by read transactions be consistent with any cached copies of the data. The simplest approach is for caching processors to use a write-through cache, where all processor write transactions are written to memory as well as to the cache. This technique will work for caching processors accessing physically local or external memory, and guarantees that data read from memory will be up-to-date. If the shared memory is physically located on the processor module, a copyback cache-coherence policy can be used if the processor can snoop and supply its modified data to external read transactions that reference its memory. Alternatively, cache coherency can be enforced by software flushing the processor caches to memory prior to any read transactions by noncaching masters.

It is recommended that memory locations accessed by nonglobal write transactions be snooped by caches so that the data in the caches are up-to-date. Alternatively, cache coherency can be enforced by software invalidating areas of memory prior to any write transactions by non-cache-coherent masters.

Cache-coherent states

The cache-coherency protocol consists of several states that are defined by the current and previous states of /CM2 and /CM1 and the type of cache-coherent transaction requested by the master. The cache-coherent transactions and their /CMx signal line encodings are summarized in Table 3-7. All cache-coherent transactions are identified by the assertion of /CM2 after /CM2 was previously unasserted. The state of /CM1 during the previous cycle identifies the actual start of the cache-coherent transaction and subsequent write-back and retry transactions.

■ **Table 3-7** Cache-coherent transaction encodings

Bus cycle *n*–1		Bus cycle *n*			
/CM2†	**/CM1**	**/CM2**	**/CM1**	**/CM0**	**Transaction**
H	H (cc-start)	L	L	L	`ReadExclusive`, `AttentionExclusive`, or `WriteExclusive`
H	H (cc-start)	L	L	H	`ReadShared` or `AttentionShared`
H	H (cc-start)	L	H	L	`ReadInvalidate`, `AttentionInvalidate`, or `WriteInvalidate`
H	H (cc-start)	L	H	H	`ReadNosnoop` or `WriteNosnoop`
H	L (nosnoop)	L	—	—	Write-back or retry (nosnoop)

† For all cache-coherent transactions starting at bus cycle *n,* /CM2 must have been unasserted high during the previous bus cycle, *n* – 1, and /CM2 must be asserted low at bus cycle *n.*

The cache-coherency states are defined here.

- The cache-coherent idle cycle (cc-idle) is the inactive state of the cache-coherency protocol where /CM2 and /CM1 are both unasserted.
- The cache-coherent start cycle (cc-start) is a single bus cycle in which /CM2 is asserted during the bus cycle immediately following a cc-idle cycle. The cc-start cycle must be coincident with the /START cycle of a data-transfer or attention cycle.
- The cache-coherent snoop cycles (cc-snoop) assert /CM2 immediately after the cc-start cycle if additional time is required to snoop the transaction and check for a cache hit. As each snooping module determines the status of its cache tags, it releases the /CM2 line and asserts its cache line status code on /CM1 and /CM0.
- The cache-coherent acknowledge cycle (cc-ack) is a single bus cycle in which /CM2 is unasserted after having been asserted during the cc-start and cc-snoop cycles. The last snooping module to release the /CM2 line determines when the cc-ack cycle occurs, and the bus master samples the cache status code on /CM1 and /CM0.

During the cc-ack cycle, one or more snooping modules may have asserted /CM0 to indicate that they have a valid (shared or exclusive) copy of the data. If none of the snooping caches contain modified data (/CM1 unasserted), the protocol is considered to be in the cc-ack/cc-idle state since /CM2 and /CM1 are both unasserted and another cache-coherent transaction may begin on the next cycle. The master updates its cache line state with the "shared" status information returned on /CM0.

During the cc-ack cycle, one or more snooping modules may have asserted /CM1 to indicate that they will perform write-back transfers to return their modified data to shared memory. In addition, the master may retry the original data transfer to update its cache and cache line tags with the "shared" status information returned on /CM0 during the cc-ack cycle.

The /CMx signal lines are used to identify write-back and retry operations and to determine when modules can request bus ownership. The master asserts /CM1 immediately after the cc-ack cycle if it intends to retry the original data transaction. Snooping modules continue to assert /CM1 and also assert /CM0 immediately after the cc-ack cycle. As each snooping module becomes bus owner and performs its write-back transaction, the module releases both /CM1 and /CM0. After all snooping modules have written back their modified data to memory, /CM0 will be in the unasserted state (and /CM1 remains asserted if the master intends to retry the original data transaction).

Sensing the unassertion of /CM0, the master can then retry the original data transaction to update its cache. When the master begins the retry operation, it releases /CM1. Once /CM1 is released, other cache-coherent modules may request bus ownership.

There are several rules that govern the relationship between cache coherency and data transaction protocols.

- The cache-coherent transaction must be coincident with the data or attention-cache transaction's /START cycle.
- The cache-coherent acknowledge cycle can occur before, during, or after the data transaction's acknowledge cycle.
- A cache-coherent cc-start cycle and associated data transaction /START cycle must wait for the completion of the previous cache-coherent transaction.
- Any number of non-cache-coherent transactions may occur during a single cache-coherent transaction.

Cache-coherent masters

A cache-coherent master indicates the type of cache-coherent transactions on the /CMx lines during the cc-start cycle. After the snooping modules have snooped the transaction, the master notes the systemwide cache status on the /CMx lines reported during the cc-ack cycle. The assertion of /CM1 indicates that one or more caches have a modified copy of data that will be returned to memory using the write-back transaction. The assertion of /CM0 indicates that one or more caches may have retained a shared copy of the data.

If /CM1 was asserted during the cc-ack cycle, the master waits until all caches have completed their write-back transactions. Then the master retries the read or write data transfer originally requested during the cache-coherent start cycle.

By retrying the read data transfer of a `ReadExclusive` or `ReadShared` transaction, the master can update its cache with formerly modified data from snooping caches. By retrying the write data transfer of a `WriteExclusive` transaction, the master can write over formerly modified data from snooping caches. In an optimization for `ReadExclusive` and `ReadShared` transactions, the master can snarf the write-back transactions and thus eliminate the need to retry the original read data transfer.

At the conclusion of the cache-coherent transaction, the master updates its cache line state according to Table 3-8.

■ **Table 3-8** Cache-coherent master actions

Mode and initial state†		cc-ack status		Action and final state	
Transaction mode	**Master state**	**/CM1**	**/CM0**	**Action**	**Master state**
Exclusive	S or I	L	H	Retry/snarf	M, E, S, or I
Exclusive	S or I	H	H	None	M, E, S, or I
Shared	I	L	L	Retry/snarf	S or I
Shared	I	L	H	Retry/snarf	E, S, or I
Shared	I	H	L	None	S or I
Shared	I	H	H	None	E, S, or I
Invalidate	S or I	H	H	None	M, E, S, or I
`ReadNosnoop`	I (DMA transfer)	H	H	None	I (data read)
`ReadNosnoop`	All not M	H	H	None	S or I
`WriteNosnoop`	I (DMA transfer)	H	H	None	I (data written)
Copyback	M	H	H	None	E, S, or I
Invalidate	M, E, S, or I	L	n/a	Error	
Nosnoop		n/a	H		

M = modified, E = exclusive, S = shared, and I = invalid.

"Retry/snarf"	/CM1 asserted during cc-ack cycle; master must retry snarf.
"None"	/CM1 not asserted during cc-ack cycle; transaction completed.
"All not M"	No cache has a modified copy of the data, or read-only data.
"Copyback"	A `WriteNosnoop` transaction used to return modified data to memory and free the cache line for later use.
"Error"	/CM1 and /CM0 should not be asserted during the cc-ack cycle.

† The "initial" state of the master cache line is when the master becomes bus owner and begins the cache-coherent transaction. The cache line may have been invalidated while arbitrating for the bus.

Arbitration by cache-coherent modules

Cache-coherent modules use the existing bus arbitration protocol to acquire the bus ownership. They also use an additional protocol and signal line, /CBUSY, to ensure that all cache-coherent modules have fair access to the bus.

To perform a cache-coherent transaction, a cache-coherent module requests bus ownership by driving the /RQST and /CBUSY lines asserted. The module continues to assert both signals until it wins the arbitration contest and asserts /START. At this point, the module can begin a cache-coherent transaction. Before doing so, it may be that the module must perform a write-back transaction in response to another module's cache-coherent transaction. The cache-coherent module may also have to relinquish bus ownership to permit other snooping modules to perform their write-back transactions or to permit another master to retry its original data transaction.

Assuming that the cache-coherent module can begin its own transaction, it releases /RQST and /CBUSY. If it must perform a write-back transaction or relinquish bus ownership, the module releases /RQST when the write-back transaction is complete, or the bus ownership has been relinquished. The module continues to assert /CBUSY until it actually starts its originally intended cache-coherent transaction.

This protocol ensures that all cache-coherent transactions requested during the same bus cycle will be serviced before any new cache-coherent requests can be made. The protocol guarantees fair access to the bus by all cache-coherent modules and is similar to the protocol used by /RQST to guarantee fair access to all modules.

Nonaligned microprocessor accesses

The MC68020, MC68030, and MC68040 bus interfaces allow accesses that do not fall into natural NuBus transactions. For example, a microprocessor program can request a read of a NuBus word at an odd-numbered location. This cannot be performed in a single NuBus transaction since it falls across word boundaries.

The interface between a Macintosh computer and NuBus always translates an aligned request into its counterpart on NuBus. However, that interface also provides support for all nonaligned microprocessor accesses. On 68020-based and 68030-based machines, the dynamic bus-sizing facilities of the microprocessor are used. However, on 68040-based machines, the microprocessor will perform multiple aligned transfers to move a non-aligned access. The computer-to-NuBus interface responds to off-boundary microprocessor requests with /DSACK (data transfer and size acknowledge) signals that tell the processor that the bus is only 16 bits wide. This causes the processor to make several cycles to fulfill the original request, using an incremented address and decremented size. For each subsequent cycle, the NuBus interface generates an appropriate transaction until the entire request is complete.

Nonaligned reads

Nonaligned reads are mapped into NuBus word (32 bit) reads. This feature provides the required data in the fewest NuBus transactions. On 68020-based and 68030-based computers, some nonaligned requests generate two NuBus cycles. For example, a NuBus word read of $Fs00 0001 generates a word read to $Fs00 0000 and a byte read to $Fs00 0004. The "extra" data provided by the word read is ignored. The reason for the second read being a byte read instead of another word read is that the processor asks for 1 byte to $Fs00 0004, which is a natural NuBus transaction.

On 68040-based Macintosh computers, a NuBus word read of $Fs00 0001 would generate three accesses: a byte access to $Fs00 0001, a word access to $Fs00 0002, and a byte access to $Fs00 0004.

Nonaligned writes

Nonaligned writes are supported by breaking the processor request into pieces that can be executed by the NuBus. For example, a NuBus word write to $Fs00 0001 would be performed in three pieces: a byte write to $Fs00 0001, a NuBus halfword write to $Fs00 0002, and a byte write to $Fs00 0004.

Data caching

The MC68030 microprocessor used in some Macintosh computers includes a feature called **data caching.** To support this feature, RAM-like cards should always supply all 32 bits, regardless of the NuBus request. For example, if a NuBus request is presented for a byte, the card should present data for all 4 bytes in the NuBus word.

Note that the caching of data can be controlled by software; that is, some address spaces can be declared as noncacheable. (NuBus space is always noncacheable.) To declare a RAM address space as noncacheable, your driver should use the `LockMemory` function. This function is described in more detail with the memory management information in *Inside Macintosh.* Any card that is not capable of supporting a full 32-bit read must have its corresponding driver software set up the caching control appropriately.

A similar caveat concerns the nonaligned cases: if a card cannot support a full 32-bit read, the software must ensure that only appropriately aligned and sized operations are requested.

Compliance categories

You may design cards that conform to the NuBus specification but do not support all NuBus features. Masters and slaves do not need to support all transfer types. Any combination of 8-, 16-, and 32-bit single data transfers, with the card acting as either master or slave, is allowable. Masters need not support all possible block transfers. However, slaves must support all block-transfer lengths if they support block transfer at all.

The decisions about how nonaligned accesses work and the rules for data caching have been made to provide the highest performance for 32-bit-wide cards. These cards may have all the necessary logic and bus transceivers to support these rules.

NuBus slot cards may be dumb. It is not required that all devices respond with an error status code for transfer types that they do not handle; it is acceptable to merely respond with an /ACK assertion.

Such dumb cards must be managed only by device drivers that are designed to communicate with them appropriately. One of the functions of the declaration ROM is to provide indications of the capabilities of the card. (The declaration ROM is described in Chapter 8, "NuBus Card Firmware.")

Driver-supported cards are those that are accessed indirectly via a software driver. You can write the driver to manage any idiosyncrasies of the card. For these types of cards, you have relative freedom in the tradeoffs you make in the design of the hardware because you can write the driver software to accommodate them.

Peer cards are cards that are designed to execute code that is not specialized to the card—for example, two cards that execute cooperating processes to solve a problem. These cards must be more general in their hardware design, because the code that executes on them assumes no restrictions in types of access, size of data operands, and so forth.

In general, peer cards must be designed to support the maximum size of transfer that any of their peers are capable of supporting. In particular, a peer card that is designed to cooperate with the MC68020, MC68030, or MC68040 microprocessor on the main logic board of a Macintosh computer must properly handle 32-bit (NuBus word) transfers. If such a card contains, for example, an MC68000 and has a local bus that is naturally 16 bits wide, the card must provide the hardware support in its NuBus interface to handle such 32-bit transfers. This would involve doing two local bus cycles for each NuBus word request.

A card with an MC68000 processor must make two NuBus halfword requests to satisfy an access to a NuBus word quantity (for example, a pointer value). A Macintosh computer properly responds to these two requests. The same instruction when executed by the MC68020, MC68030, or MC68040 microprocessor makes one NuBus word request. If the card with the MC68000 does not respond with the correct 32-bit quantity, the program obviously does not execute correctly.

You should clearly indicate in the card's documentation exactly which kind of card it is and what types of accesses it supports.

Memory devices, however, must support all transfer types except for block transfer; the devices should always respond with all 32 bits in the addressed NuBus word. This rule allows RAM cards to be used as if they were on-board RAM in order to support nonaligned transfers, 68020 bit-field instructions, 68030 caching, and so forth.

Chapter 4 **NuBus Arbitration**

This chapter discusses how the bus master is selected from among the several cards likely to be competing for bus mastership, and how all the other cards desiring service are accommodated.

Arbitration overview

The NuBus fair arbitration mechanism differs from strict priority arbitration in that it prevents "starvation" of cards and distributes access to the bus evenly.

Arbitrate 3 to arbitrate 0 (/ARB3–/ARB0) are open-collector binary-coded lines driven by contenders for the bus. They are used by the distributed arbitration logic to determine bus mastership.

Bus request (/RQST) is an open-collector line driven low by contenders for the bus.

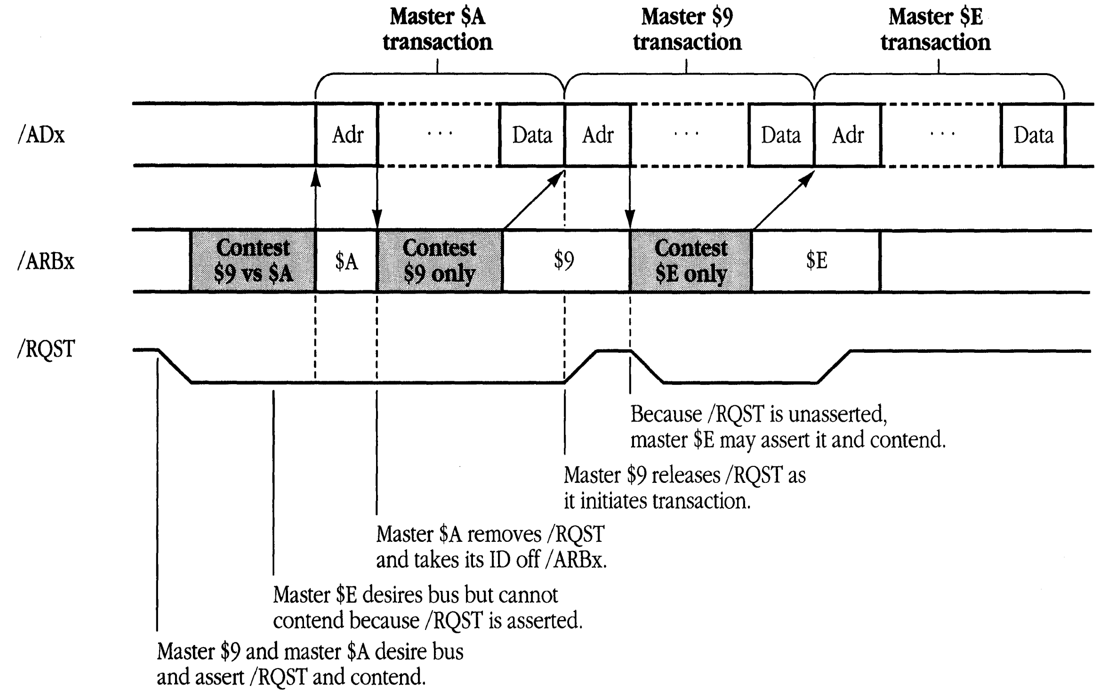
During arbitration, one or more cards contend for control of the NuBus. Cards that desire ownership of the NuBus must first assert the /RQST line. The /RQST line may be asserted only while it is in an unasserted state. All cards that assert /RQST place their ID codes on the /ARBx lines and contend for the bus. The arbitration logic distributed among the cards determines which of the cards gets ownership of the NuBus. After two clock periods, signal transients have settled and the contest mechanism is complete. The contender with the highest ID code has its code on the /ARBx lines, has won bus ownership, and may initiate a transaction (after completion of any transactions in progress).

Presuming that the winner does not desire to lock the bus, the winning card first removes its /RQST and at the same time asserts /START (this begins a start cycle of the card's first transaction). Then, after the start cycle, the card removes its /ARBx signals and continues with the cycles required to complete the transaction.

The release of /START initiates another contest between any cards that originally requested the bus in the same clock period, but that have not yet won. These cards will be granted ownership in turn, from highest ID number to lowest ID number. The rule that /RQST must be unasserted before a card may assert it keeps other cards from participating in contests until all the original requestors have been served.

Figure 4-1 shows a situation in which cards with ID codes $9 and $A request the bus at the same clock period. Card $A wins the first arbitration contest, and then removes its request after its start cycle (when the address is shown on the /ADx lines). In the meantime, card $9 continues to assert /RQST. Card $E desires the bus as well but may not request it because the /RQST line is already asserted by card $9. Contesting against no one, card $9 wins the next contest and gains bus ownership. When card $9 releases /RQST, card $E requests, arbitrates, and wins. Note that card $9 owns the bus only after it wins a contest and the transaction in progress ends.

■ **Figure 4-1** Sample arbitration contest

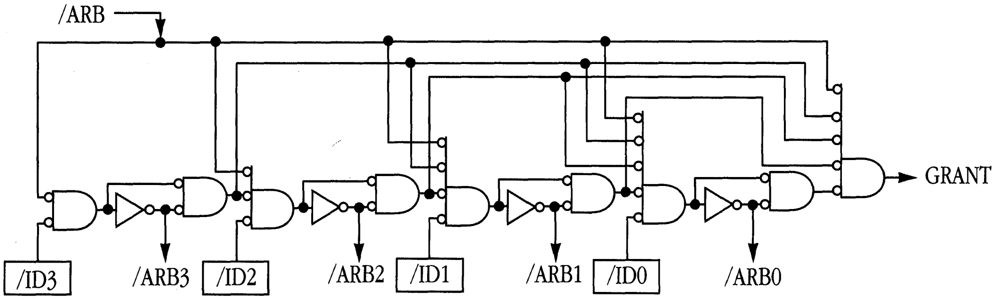


Master $A transaction
Master $9 transaction
Master $E transaction
/ADx
Adr
· · ·
Data
Adr
· · ·
Data
Adr
· · ·
Data
/ARBx
Contest $9 vs $A
$A
Contest $9 only
$9
Contest $E only
$E
/RQST
Because /RQST is unasserted, master $E may assert it and contend.
Master $9 releases /RQST as it initiates transaction.
Master $A removes /RQST and takes its ID off /ARBx.
Master $E desires bus but cannot contend because /RQST is asserted.
Master $9 and master $A desire bus and assert /RQST and contend.

Arbitration logic mechanism

When a bus contest occurs, each card drives the /ARBx lines with its unique ID code and then releases the /ARBx lines if it detects higher ID codes than its own on the /ARBx lines. One possible implementation of this arbitration logic is diagrammed in Figure 4-2, for illustrative purposes only.

■ **Figure 4-2** Typical bus arbitration logic



Note that the /ARBx lines are bused common to all cards, but the /IDx lines present a unique binary code to each card slot. The signals /ARB and GRANT are card signals, not NuBus signals: /ARB is an input to the arbitration logic that indicates whether the card is contending for the bus, and GRANT is an output that indicates whether the /ARBx lines currently match this card's /IDx lines. The following logic equations approximate how the arbitration logic on any given card works:

/ARB3 = /ID3 • /ARB
/ARB2 = /ID2 • /ARB • (/ID3 + ARB3)
/ARB1 = /ID1 • /ARB • (/ID3 + ARB3 • (/ID2 + ARB2)
/ARB0 = /1D0 • /ARB • (/ID3 + ARB3) • (/ID2 + ARB2) • (/ID1 + ARB1)

where • is logical AND, + is logical OR, and ARBx is the logical complement of /ARBx.

According to these equations, after a short delay (arbitration period) the /ARBx lines will equal the ID code of the highest-priority contender, that is, the contender with the largest integer for its ID code. See Appendix D for the PAL listing labeled (ARB2), NuBus arbitration logic; implementation of these equations accomplishes the desired arbitration.

◆ *Note:* The signal names /ARB and GRANT are written here with capital letters, consistent with the convention used in this book, but the Texas Instruments NuBus documentation uses /arb and grant, respectively.

Arbitration timing overview

The details of arbitration timing are covered in Chapter 5, "NuBus Card Electrical Design Guide." Arbitration events generally occur on driving edges and sampling edges, synchronous to the system clock, with the same timing as the basic address/data, control, and utility signals. For example, /RQST may be asserted on a particular driving edge only if it is seen to be unasserted on the previous sample edge. However, the /ARBx lines differ from all other NuBus signals in that their assertion timing is specified from the sample edge of the bus clock. See Figures 4-3 and 5-2.

Arbitration contests last two clock periods by definition. On the second sampling edge after a contest starts, all contenders sample their internal GRANT signal. The highest-priority contender will find its GRANT signal asserted. The winner may now take control of the bus and assert /START on the next driving edge (25 ns after the contest's second sampling edge) if the bus isn't in use.

If the bus is in use, the new winner asserts /START on the driving edge immediately after the next sample edge where the current transaction's /ACK is asserted. The new winner continues to assert its ID code on the /ARBx lines throughout the start cycle of its first transaction. This facilitates bus lock detection and bus diagnostics.

▲ **Warning** If there are many NuBus masters active at the same time, the CPU, especially in the Macintosh IIfx, can be given too little time to refresh memory. This problem is actually caused by the "fair" NuBus arbitration scheme; no priority is given the CPU in its bid for NuBus access. This can cause the computer to slow down tremendously. On most Macintosh computers, the symptom of this problem will be extremely slow or nonexistent updates to the screen. ▲

Locking

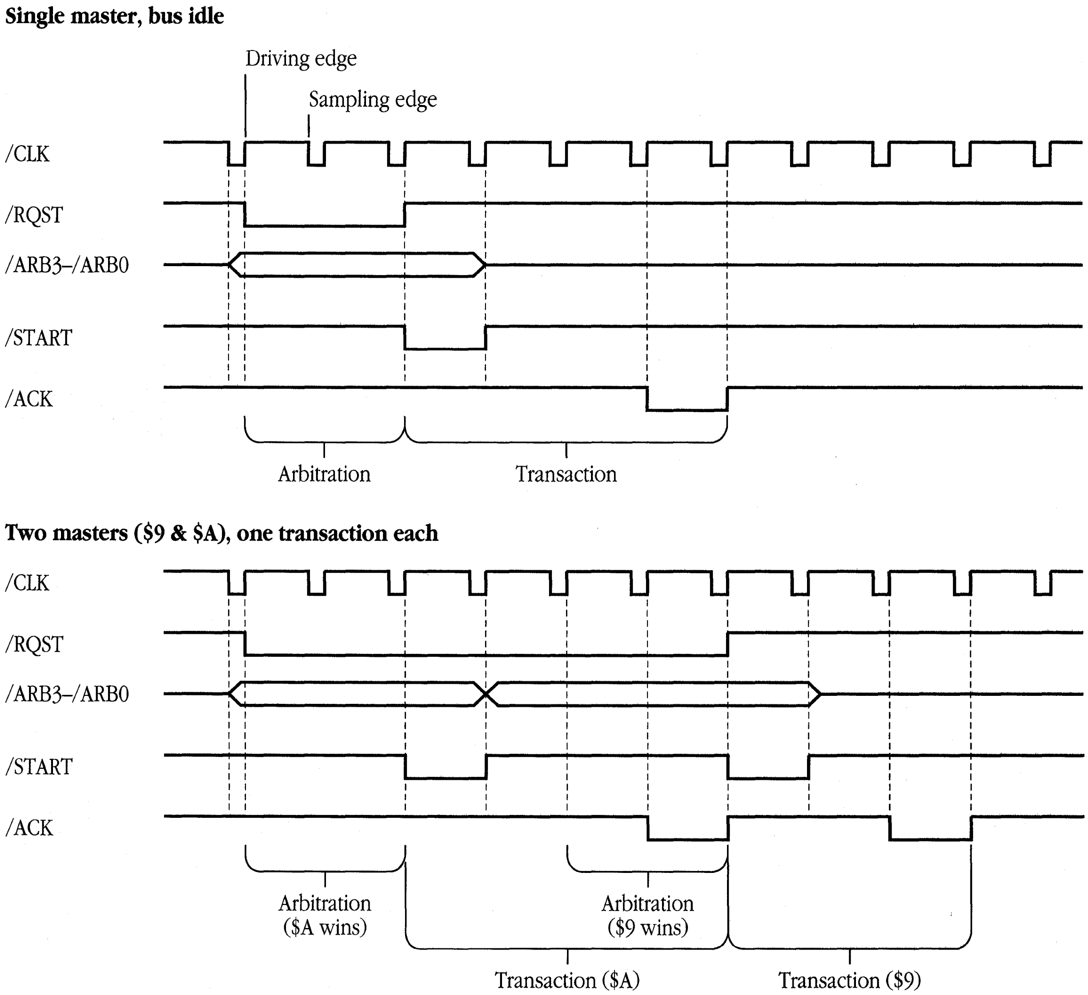
Although cards generally use the bus for a single transaction before allowing another requesting card to become bus master, sometimes the bus must be held locked in an extended tenure. For some local processor operations, it may be necessary to prevent any NuBus requests from interfering with the access of the processor to its local bus.

This might be the case, for example, when the processor is doing a floppy disk transfer, which is inherently time critical. Such a processor must have some mechanism (for example, a bus lock line) for locking itself, and its local bus, from NuBus intrusion. This type of locking is called **bus locking.**

Another example of locking to prevent interference is an indivisible test-and-set operation performed in a multiprocessor environment; this type of locking is called **resource locking.**

△ **Important** The bus must not be held in a locked condition for more than four transactions at a time. △

■ **Figure 4-3** NuBus arbitration and transaction timing, single master and two masters



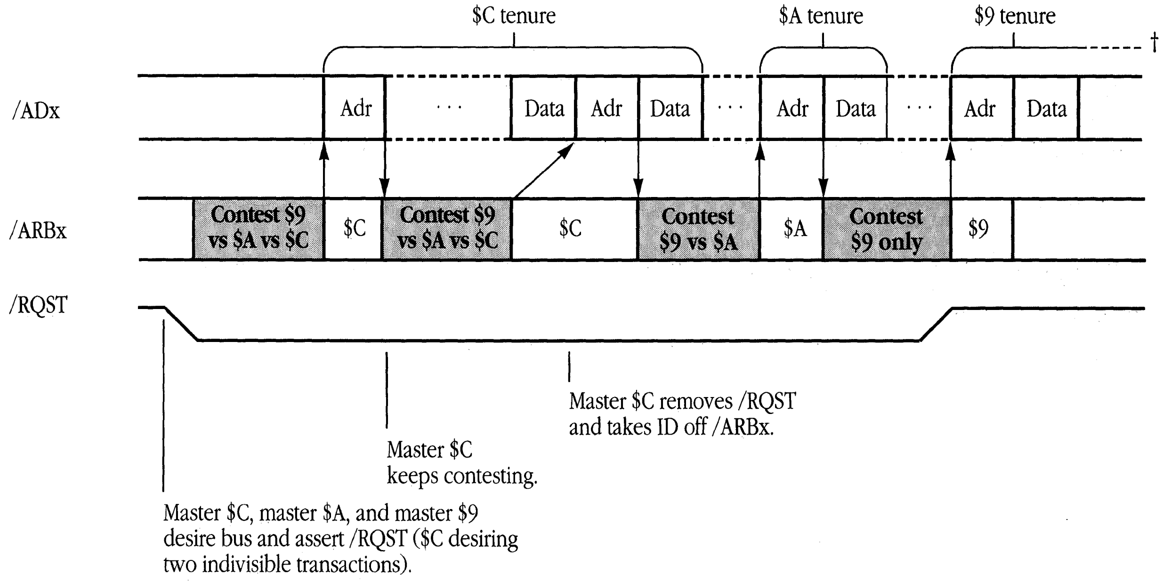
Bus locking

Bus locking requires no added mechanism. To lock the bus, a master simply continues to request (by keeping the /ARB lines driven with its ID code) and contend (continuing to assert /RQST). Because it has the highest ID code of those cards present, it wins subsequent contests. Figure 4-4 shows an example in which card $C locks the bus for two transactions. Fairness in arbitration depends upon cards not locking the NuBus unless required and locking it only for the shortest required tenure.

Any card or software that uses extended-tenure bus locking should clearly specify in the documentation for the product the maximum number of bus cycles allowed.

Bus locking can be valuable for optimizing NuBus transfers. During a NuBus transfer, it is not possible to determine when a transaction will complete because of a variance in NuBus interrupt latency. The interrupt latency can be affected by such things as floppy disk accesses and slot interrupts. The bus, therefore, can be locked for a small set of transactions, then unlocked for rearbitration. It order to allow fairness, you should lock the bus for as short a time as possible. It is also important that you provide sufficient buffering on your NuBus card to allow for the variance in interrupt latency.

■ **Figure 4-4** Sample bus lock


$C tenure
$A tenure
$9 tenure
†
/ADx
Adr
. . .
Data
Adr
Data
. . .
Adr
Data
. . .
Adr
Data
/ARBx
Contest $9 vs $A vs $C
$C
Contest $9 vs $A vs $C
$C
Contest $9 vs $A
$A
Contest $9 only
$9
/RQST
Master $C removes /RQST and takes ID off /ARBx.
Master $C keeps contesting.
Master $C, master $A, and master $9 desire bus and assert /RQST ($C desiring two indivisible transactions).

† Tenure continues until another card asserts /RQST and the next arbitration contest commences.

Resource locking

Resource locking is initiated by the bus owner driving both /START and /ACK to commence an **attention-resource-lock cycle;** this alerts all cards that a bus and resource locked transaction is occurring. The bus lock is maintained as described in the previous section. A bus owner that issues an attention-resource-lock cycle as the first cycle of a bus tenure must conclude that tenure with an attention-null cycle to inform all cards that the tenure is complete.

Access to a resource must be controlled when that resource is accessible by both a local processor and the NuBus. One example of such a shared resource is a dual-ported RAM. Another, more specific, example is found in Macintosh computers, where the NuBus interface circuitry uses the local processor bus to access the shared resource, RAM, as shown in Figure 4-5.

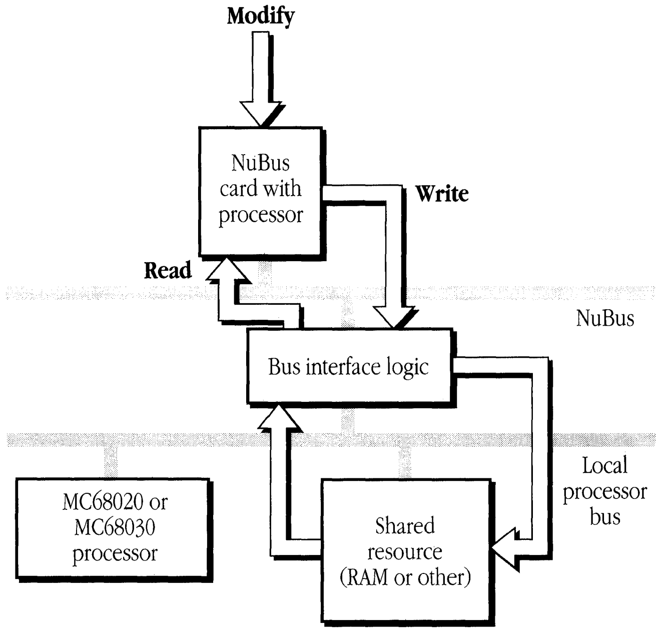
All cards that have shared resources capable of being locked must monitor the NuBus for an attention-resource-lock cycle and must record the occurrence. A card does not have to react to the occurrence of a bus tenure starting with an attention-resource-lock cycle unless it is addressed during that tenure; this allows multiple resources to be alerted and locked during a single bus tenure.

Figure 4-5 may be helpful in discussing an indivisible bus operation. For example, suppose the processor on the NuBus card is instructed to perform a read-modify-write cycle to the RAM as part of executing a TAS (test and set) instruction. The NuBus card contends for and wins bus ownership, then initiates an attention-resource-lock cycle. On the Macintosh IIcx, Macintosh IIci, Macintosh IIsi, Macintosh IIfx, Macintosh Quadra 700, and Macintosh Quadra 900, the NuBus interface controller automatically performs an attention-resource-lock cycle before doing a read-modify-write cycle, and an attention-null cycle afterward.

The state machines in the BIU respond to the attention-resource-lock cycle by setting a flag. This flag indicates that if the RAM-shared resource is accessed by the processor on the NuBus card, the BIU will lock the processor bus. The local processor will then be unable to access the RAM and thereby interfere with the indivisible read-modify-write of a data structure by the NuBus processor. Any bus owner that is programmed to perform an indivisible bus operation should lock resources on any slaves to be addressed during that operation, as well as locking any bus that provides an alternative path to those resources. All cards should release the locked status when an attention-null cycle occurs.

A card is not *required* to provide locking of its local resources; it may do so on some resources and not on others. Reliable TAS instructions may only be done on resources that can be locked.

■ **Figure 4-5** Read-modify-write indivisible bus operation


Modify
NuBus card with processor
Write
Read
NuBus
Bus interface logic
Local processor bus
MC68020 or MC68030 processor
Shared resource (RAM or other)

Bus parking

A bus master that has released /RQST is considered **parked** on the bus and may use it at any time (without rearbitration) until another card asserts /RQST. When /RQST is finally asserted by another requester, the parked bus master finishes its current transaction and relinquishes the bus to the new winner without commencing another transaction. Bus parking reduces the average time to acquire the bus in systems with a small number of contenders.

◆ *Note:* A bus owner is not allowed to go from a parked condition into a bus-locked series of transactions without submitting to arbitration by asserting /RQST.

Chapter 5 **NuBus Card Electrical Design Guide**

This chapter gives the electrical specifications and timing requirements for NuBus cards, including power requirements, connector pin assignments, a power budget, and timing diagrams. Refer to Appendix A for guidelines on electromagnetic interference (EMI), heat dissipation, and product safety.

Electrical requirements

This section provides the detailed electrical information that you need to design a NuBus expansion card.

Logical and electrical state relationships

All NuBus signals are active when low. The relationship between logical states and electrical signal levels for all NuBus lines is shown in Table 5-1.

■ **Table 5-1** Logical state definitions

Logical state	Electrical signal level
H (unasserted)	> 2.0 V at the receiver
L (asserted)	< 0.8 V at the receiver

DC and AC specifications for line drive

This section provides the drive requirements and the load allowance for each of the NuBus lines. These lines can be divided into five basic types based on their electrical drive and load characteristics:

- clock (/CLK)
- address/data (/ADx, /SP, /SPV)
- control (/START, /ACK, /TMx)
- open collector (/RESET, /RQST, /ARBx, /NMRQ, /CBUSY†, /CBx†)
- power control (/PFW)

† These signals were introduced in the NuBus '90 specification and are defined only in the Macintosh Quadra 700 and the Macintosh Quadra 900.

Table 5-2 lists the specifications for these line (signal) types, from a NuBus card's point of view. The columns labeled *drive* indicate the minimum requirements for card outputs, while those labeled *load* specify the maximum load that may be presented by card inputs. Negative currents indicate flow out of a node (sourcing), and positive currents indicate flow into a node (sinking).

■ **Table 5-2** NuBus line drive requirements and load allowances

	AC drive		DC drive		AC load	DC load	
Signal type	I_{PD} (min), mA	I_{PU} (min), mA	I_{OL} (min), mA	I_{OH} (min), mA	C_L (max), pF	I_{IL} (max), mA	I_{IH} (max), mA
Address/data	80	40	24	–12 mA @3.2 V	18	–0.5	0.1
Clock†	90	50	60	–30	18	–1.4	0.1 from driver
Control	80	40	24	–12 mA @3.2 V	18	–0.5	0.1
Open collector	80	N/A	60	N/A	18	–0.625	0.1
Power control (/PFW)‡							

† Supplied by the Macintosh computer.

‡ The source of /PFW must be capable of sourcing 20 mA at 3 V for 2 seconds when driving /PFW high to turn the computing system on. See the next section.

The column headings in Table 5-2 have the following meanings:

I_{PD} Transient pull-down current, required for one T_{pd} (NuBus delay period) whenever the driver changes from unasserted to asserted.

I_{PU} Transient pull-up current, required for one T_{pd} whenever the driver changes from asserted to unasserted.

I_{OL} Low-output drive current available at 0.5 V.

I_{OH} High-output drive current available at specified voltage.

C_L Capacitive load per slot.

I_{IL} DC low-level input current.

I_{IH} DC high-level input current.

◆ *Note:* Each NuBus card input can present an AC load of up to 18 pF to the computer's main logic board. This includes 2 pF for the NuBus connector and 16 pF for the card's trace capacitance plus the input capacitance of all devices connected to that trace. Also, the load presented by NuBus card inputs (and tristated outputs) affects those NuBus signals as seen by the computer's main logic board and by any other installed NuBus cards. To minimize NuBus signal degradation, it is best to buffer all card inputs as close to the NuBus connector as possible, and to limit each signal to one LS load. This is especially important for the NuBus clock signal, which, because of its critical timing and high frequency, is easily damaged by the loading effect of a NuBus card. For additional helpful design hints, see Appendix A, "EMI, Heat Dissipation, and Product Safety Guidelines."

/PFW interaction with the power supply

The /PFW signal is intended to serve two purposes:

1. To allow the power supply to be turned on and off by a low-voltage signal that can be controlled by the logic board (or expansion card) circuitry and hence by software.
2. To allow the power supply to warn the computer of an impending power loss.

When /PFW is held between 3.0 and 6.8 V for at least 1.5 seconds, the power supply turns on and the computer begins operating. Once the power supply turns on, its own +5V output holds /PFW high so it can continue operating. If /PFW is pulled below 0.6 V, the power supply will turn off; /PFW should be *held* below 0.6 V until the computer completely shuts down. If some fault condition (such as AC line failure) causes the power supply to turn off, the power supply will pull /PFW low at least 2 ms before the DC outputs fail.

There are many issues that restrict the circuitry that can be connected to /PFW. Here are a few cautions and tips:

- The /PFW voltage may be greater than the +5V bus voltage for a second or two when the computer is turned on.
- If /PFW is fed into a gate input, any internal diodes to the +5V (or any other power) bus may prevent the computer from turning on because /PFW goes high before the power supply outputs bring the power buses up to rated voltage.
- No pull-up may be added to the /PFW line or else there may be difficulty in turning off the computer.
- Any circuitry connected to /PFW must present a high impedance when the power is removed or it may prevent the computer from turning on and drain the battery. Likewise, such circuitry must present a high-impedance load during normal operation to prevent contention with other drivers of /PFW. The only time additional circuitry should present a low-impedance load to the /PFW line is when it is intentionally and temporarily controlling the /PFW signal.

NuBus connector pin assignments

Table 5-3 gives the pin assignments for NuBus connectors on most of the Macintosh computers with the NuBus interface. The order of the rows is given as viewed from the front edge of the card. The NuBus '90 specification has changed several of these pin assignments. Table 5-4 lists the connector pin assignments for NuBus connectors on the Macintosh Quadra 700 and Macintosh Quadra 900. Table 5-5 lists and describes the new signals defined in NuBus '90 and included in the Macintosh Quadra family.

- **Table 5-3** Connector pin assignments

Pin	Row A	Row B	Row C
1	–12V	–12V	/RESET
2	‡	GND	‡
3	/SPV	GND	+5V
4	/SP	+5V	+5V
5	/TM1	+5V	/TM0
6	/AD1	+5V	/AD0
7	/AD3	+5V	/AD2
8	/AD5	†	/AD4
9	/AD7	†	/AD6
10	/AD9	†	/AD8
11	/AD11	†	/AD10
12	/AD13	GND	/AD12
13	/AD15	GND	/AD14
14	/AD17	GND	/AD16
15	/AD19	GND	/AD18
16	/AD21	GND	/AD20
17	/AD23	GND	/AD22
18	/AD25	GND	/AD24
19	/AD27	GND	/AD26
20	/AD29	GND	/AD28
21	/AD31	GND	/AD30
22	GND	GND	GND
23	GND	GND	/PFW
24	/ARB1	†	/ARB0
25	/ARB3	†	/ARB2
26	/ID1	†	/ID0
27	/ID3	†	/ID2
28	/ACK	+5V	/START
29	+5V	+5V	+5V
30	/RQST	GND	+5V
31	/NMRQ	GND	GND
32	+12V	+12V	/CLK

† These pins are connected but not supplied with the –5.2 V described in the Texas Instruments NuBus specification. This voltage could be supplied by a card, in which case –5.2 V would be available to all cards.

‡ These pins are reserved in the IEEE 1196 standard and are grounded in all Macintosh computers with the NuBus interface except for the Macintosh Quadra 700 and Macintosh Quadra 900.

■ **Table 5-4** NuBus '90 connector pin assignments

Pin	Row A	Row B	Row C	Pin	Row A	Row B	Row C
1	–12V	–12V	/RESET	17	/AD23	GND	/AD22
2	SB0	GND	SB1	18	/AD25	GND	/AD24
3	/SPV	GND	+5V	19	/AD27	GND	/AD26
4	/SP	+5V	+5V	20	/AD29	GND	/AD28
5	/TM1	+5V	/TM0	21	/AD31	GND	/AD30
6	/AD1	+5V	/AD0	22	GND	GND	GND
7	/AD3	+5V	/AD2	23	GND	GND	/PFW
8	/AD5	/TM2	/AD4	24	/ARB1	CLK2X	/ARB0
9	/AD7	/CM0	/AD6	25	/ARB3	STDBYPWR	/ARB2
10	/AD9	/CM1	/AD8	26	/ID1	/CLK2XEN	/ID0
11	/AD11	/CM2	/AD10	27	/ID3	/CBUSY	/ID2
12	/AD13	GND	/AD12	28	/ACK	+5V	/START
13	/AD15	GND	/AD14	29	+5V	+5V	+5V
14	/AD17	GND	/AD16	30	/RQST	GND	+5V
15	/AD19	GND	/AD18	31	/NMRQ	GND	GND
16	/AD21	GND	/AD20	32	+12V	+12V	/CLK

△ **Important** The eight lines that were connected to the –5.2V signals in the original NuBus specification are now used for new features on the Macintosh Quadra 700 and the Macintosh Quadra 900. Many older NuBus cards connect those eight lines together; the presence of such a card in the Macintosh Quadra 700 and Macintosh Quadra 900 will disable the new features of all installed NuBus cards that use those lines. All the other features of both the old and new cards will operate normally. △

■ **Table 5-5** NuBus '90 signals on the Macintosh Quadra–family NuBus connectors

Pin number	Signal name	Function
A2	SB0†	High-speed serial bus, defined in P1394 standard.
B8	/TM2	New transfer mode: requests double-speed transfer.
B9	/CM0†	Controls cache-coherency operations.
B10	/CM1†	Controls cache-coherency operations.
B11	/CM2†	Controls cache-coherency operations.
B24	/CLK2X	Synchronizes double-speed block transfers.
B25	STDBYPWR	Small current at +5 V when main power is off; enables a card to turn on main power by asserting /PFW signal. This signal is defined only on the Macintosh Quadra 900 and not on the Macintosh Quadra 700.
B26	/CLK2XEN	If not connected to other NuBus '90 signals, this line enables /CLK2X driver.
B27	/CBUSY	Used with cache-coherency operations.
C2	SB1†	High-speed serial bus, defined in P1394 standard.

† These signals are not driven or monitored by circuits in the Macintosh Quadra 700 or the Macintosh Quadra 900.

Power supply specifications

Three voltages are specified on the NuBus: +5 V, +12 V, and –12 V. These voltages are listed in Table 5-6 with their specifications.

■ **Table 5-6** Power supply specifications

Source label	Nominal value, V	Tolerance from nominal, %	Combined line and load regulation, %	Maximum ripple, mV (peak–peak)
+5	5	±3	0.3	50
+12	12	±3	0.3	75
–12	–12	±3	0.3	75

NuBus power budget

You can determine the maximum current available to any NuBus card by dividing the maximum current available to the entire NuBus by the number of NuBus slots. For example, since a Macintosh II, Macintosh IIx, and Macintosh IIfx all have six NuBus slots, the maximum current available to any one NuBus card is one-sixth of that available to the entire NuBus. And since a Macintosh IIcx and a Macintosh IIci have only three slots, the maximum current available to any one NuBus card is one-third of that available to the entire NuBus. Worst case analysis for a fully loaded Macintosh computer, with equal current allocation to each of the slots, yields the recommendations in Table 5-7. A similar analysis, starting with the maximum capacitance for which the power supply operates reliably and subtracting the maximum capacitance on the main logic board, yields the card filter capacitance recommendations in the table.

◆ *Note:* The maximum current available to the entire NuBus in the Macintosh IIcx or Macintosh IIci computer is one-half of the maximum current available to the entire NuBus of a Macintosh II, Macintosh IIx, or Macintosh IIfx computer. Therefore, the calculated maximum current allocation to each of the three slots in a Macintosh IIcx or Macintosh IIci is the same as that shown in Table 5-7.

■ **Table 5-7** Recommended current and capacitance limits for a NuBus card

Nominal power supply value, V	Recommended maximum current per card (slot), A (continuous)	Recommended maximum capacitance per card, µF
+5	2.0	1513
+12	0.175	536
–12	0.150	698

◆ *Note:* The current analysis assumes a hard disk (1.8 A rms max) and two floppy disk drives (0.2 A typical) internal to the computer; if you choose to develop a card that exceeds these recommendations, you should make the end user aware of any limitations imposed on the system configuration.

The recommendations for maximum card capacitance are actual (not nominal) capacitance. You must allow for the capacitance tolerances of the particular capacitors being used in order to stay below the recommended maximum.

The power allowed in all Macintosh computers except the Macintosh Quadra 900 is 13.3 W per NuBus slot. The power supply in the Macintosh Quadra 900 is designed to provide additional current on the +5V outputs for the NuBus slots, compared with other Macintosh computers. The Macintosh Quadra 900 has enough power to support a total of two 25 W cards and three 15 W cards. The total power budget for NuBus cards in the Macintosh Quadra 900 shall not exceed 95 W.

If the amount of power used by NuBus expansion cards exceeds the total power budget, the Macintosh computer cannot be booted. During startup, the power supply attempts to turn itself on but cannot, and it continues the attempt over and over. When the computer is in this state, you must unplug it and remove the offending expansion cards.

However, some NuBus cards may inherently require more power. If your card contains a processor or a large amount of RAM, the card will probably need more power than is allowed for each expansion card. In the rare case when you do need to consume the power of multiple slots, you must make sure that the slot or slots adjacent to your card are not used. While there may be many ways to prevent the installation of an adjacent card, three possible solutions are provided here.

To prevent installation of an expansion card in an adjacent slot, you could create a mechanical barrier attached to your expansion card. Alternatively, you could design your NuBus expansion card as a multiple-card implementation. The NuBus cards could be connected via an internal bus, using ribbon cables or another type of connector. As a third suggestion, you could provide slot covers with your card. You must instruct the user to install slot covers over the necessary adjacent slots and warn them that they could damage their computers if the slot covers are not installed.

While all three suggestions solve the problem, there is one major drawback for the first two suggestions: if the power budget for future Macintosh computers changes, your card may no longer exceed the per slot power allocation. At that point, you may be wasting space and available NuBus slots. The third suggestion avoids this potential waste, as the slot covers would simply not be installed.

▲ **Warning** It is important that the 13.3 W power allocation not be exceeded for NuBus expansion cards in the Macintosh IIsi. Because the Macintosh IIsi has only one expansion slot, you cannot "borrow" excess power from other slots that may not be filled. Because the power supply in the Macintosh IIsi is designed to drive only a single card, however, a NuBus card that consumes more power than it is supposed to may damage itself and possibly the Macintosh IIsi. ▲

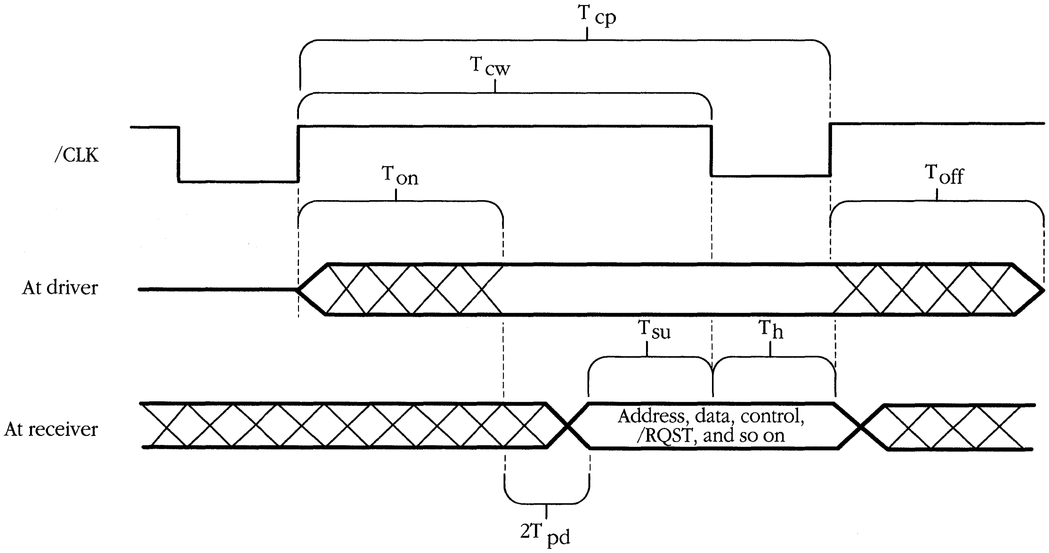
Timing requirements

To meet the following timing requirements, you must pay careful attention to card construction practices. You must provide adequate design and manufacturing margins so that cards manufactured by you and other developers may be interchangeably inserted in any Macintosh computer with the NuBus interface and all communicate with each other and the processor on the main logic board.

Utility and data-transfer timing

Figure 5-1 shows the clock, control, and address/data timing relationships during data transfers. Table 5-8 lists the bus timing specifications for these signals. Control and address/data signals are changed on the rising edge of /CLK and sampled on the falling edge of /CLK. This timing gives protection from bus skew.

■ **Figure 5-1** Data-transfer timing diagram



T_{cp}	Clock period
T_{cw}	Clock width
T_{on}	Turn-on time at driver
T_{off}	Turn-off time at driver
T_{su}	Setup time at receiver
T_h	Hold time at receiver
$2T_{pd}$	NuBus delay

■ **Table 5-8** Data-transfer timing parameters

Parameter	Description	Minimum, ns	Maximum, ns
T_{cp}	Clock period†	99.99	100.01
T_{cw}	Clock width	73	77
T_{on}	Turn-on time	0	35
T_{off}	Turn-off time	0	35
$2T_{pd}$	NuBus delay		
	(16 loaded slots)	—	17
	(8 loaded slots)	—	10
T_{su}	Setup time	21	—
T_h	Hold time	$T_{cp} - T_{cw}$	—

† This clock period is the average period over a 1-second interval. Jitter must be kept low enough to avoid violations of other parameters.

Setup, hold, and other times are defined at the card-to-NuBus connectors. All card-internal delays must be taken into account while providing for the times specified in the table.

Arbitration timing

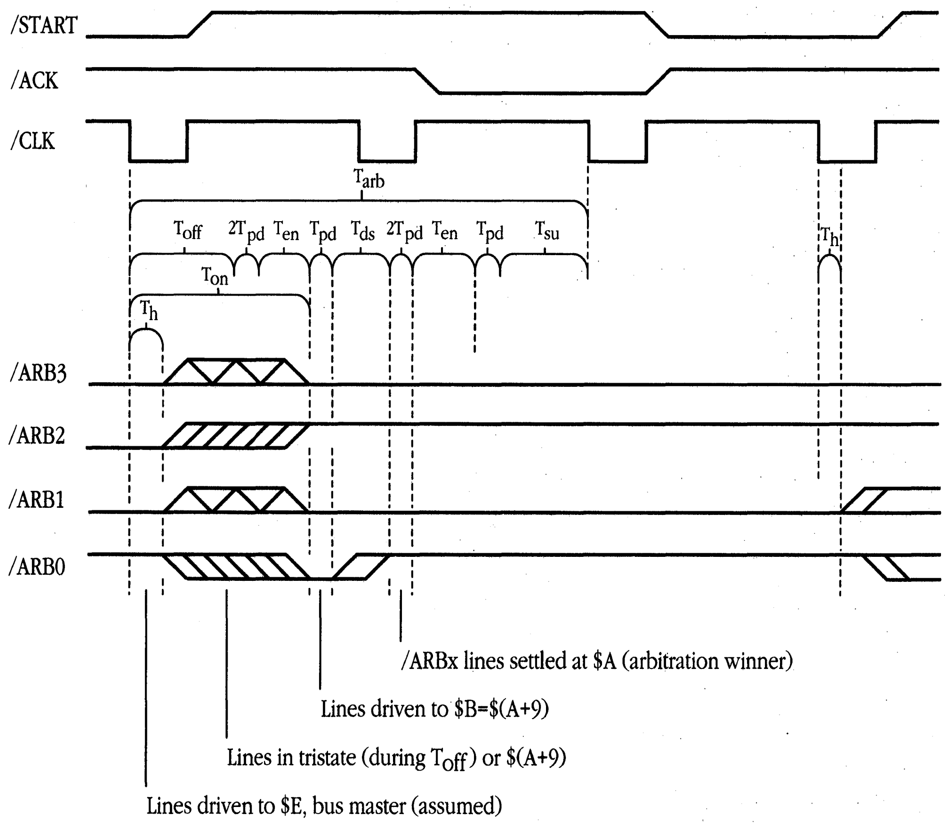
Refer to Chapter 4, "NuBus Arbitration," for a description of the arbitration process. The timing for the /ARBx signals is not the same as the timing of the data-transfer signals. Arbitration begins on the falling (sampling) edge of /CLK before the assertion of /RQST or, if /RQST is already active on the falling edge of /CLK, during /START. The contenders assert their respective slot IDs on the /ARBx lines. The bus contest must be settled within two cycles of /CLK following the assertion of /RQST or the negation of /START. By the end of that interval, the /ARB lines will contain the ID code of the card winning the arbitration contest.

Figure 5-2 details the /ARBx timing for an arbitration won by card $A following a /START signal initiated by card $9. See Table 5-9 for the meaning of the abbreviations used in Figure 5-2.

In the general case, contenders must wait for the preceding bus master to release the /ARBx lines before the succeeding bus arbitration can take place. Thus, the arbitration turn-on time (T_{on}) for /ARBx signals is the turn-off time of the preceding master (T_{off}), plus the bus propagation delay ($2T_{pd}$, one reflection assumed), plus the time taken to react to the change in logic levels (T_{en}).

Table 5-9 lists the timing specifications for the /ARBx lines.

■ **Figure 5-2** Detailed arbitration timing


/START
/ACK
/CLK
Tarb
Toff 2Tpd Ten Tpd Tds 2Tpd Ten Tpd Tsu
Ton
Th
Th
/ARB3
/ARB2
/ARB1
/ARB0
/ARBx lines settled at $A (arbitration winner)
Lines driven to $B=$(A+9)
Lines in tristate (during Toff) or $(A+9)
Lines driven to $E, bus master (assumed)

■ **Table 5-9** Bus arbitration timing parameters

Parameter	**Description**	**Minimum, ns**	**Maximum, ns**
T_{arb}	Arbitration time	—	200
T_{on}	Arbitration turn-on time	10	83
T_{ds}	Arbitration disable time	—	26
T_{en}	Arbitration enable time	—	26
T_{su}	Arbitration setup time	31	—
T_h	Hold time	10	—
T_{off}	Turn-off time	10	40
$2T_{pd}$	NuBus delay		
	16 loaded slots	—	17
	8 loaded slots	—	10

Chapter 6 **NuBus Card Physical Design Guide**

This chapter contains physical design guidelines for the development of NuBus expansion cards for Macintosh computers. It describes the physical characteristics, including the maximum allowable dimensions, of a Macintosh NuBus card.

◆ *Note:* The NuBus specification also specifies a much larger, triple-height card, but that card cannot be used in a Macintosh computer.

Card description

Foldout drawings in the back of the book show the pertinent design details and installation requirements of a NuBus expansion card. If you would like to develop a NuBus expansion card that will fit into more than one of the Macintosh computers, adhere strictly to the NuBus specification. Most Macintosh computers, because of their physical design, allow you to vary slightly from the NuBus standard. This, however, does not guarantee that all future Macintosh computers with the NuBus interface will vary in the same manner. If a NuBus card is developed according to the NuBus specification, it will be guaranteed to fit into the entire line of Macintosh computers with the NuBus interface.

All Macintosh computers with the NuBus interface will accommodate a standard NuBus card. Foldout 1 shows the overall dimensions and the placement of connectors on a standard NuBus card viewed from the component side. The NuBus connector is on the bottom, and the I/O connector is on the right side of the card.

◆ *Note:* Foldout 2 pertains only to the Macintosh Quadra family. The card in Foldout 2 has identical physical dimensions to the card in Foldout 1, but since a Macintosh Quadra–family computer uses the same size card for both NuBus and PDS expansion, the card in Foldout 2 is shown with two connectors, one a 96-pin NuBus connector and the other a 140-pin PDS connector. If you are designing a NuBus card, the PDS connector is omitted; if you are designing a PDS card, the NuBus connector is omitted.

Foldout 3 gives the clearance dimensions for installing a NuBus card in a Macintosh II, Macintosh IIx, Macintosh IIfx, or Macintosh Quadra 900. Foldout 4 gives the clearance dimensions for installing a NuBus card in a Macintosh IIcx, Macintosh IIci, or Macintosh Quadra 700.

▲ **Warning** The foldout drawings are from design guides used within Apple Computer. These drawings were correct at the time of publication but are subject to change. ▲

△ **Important** To make sure that your NuBus card fits and functions in all Macintosh computers, your physical design should adhere to the specifications in the IEEE publication *Standard for a Simple 32-Bit Backplane Bus: NuBus,* ANSI/IEEE Std 1196-1990. △

According to the IEEE NuBus specification, a standard NuBus expansion card must be 101.6 mm (4.0 inches) high and between 326.6 mm (12.858 inches) and 177.8 mm (7.0 inches) long. Notice that the cards shown in the foldout drawings exceed the NuBus-specified maximum length of 326.6 mm (12.858 inches). This is permitted because the physical design of the existing Macintosh computers allows a slight deviation from the NuBus standard. If you follow the NuBus card design guidelines shown in the foldouts, your NuBus cards will work in current Macintosh models; but to ensure that your card will work in all future Macintosh models, it is recommended that you adhere to the dimensional tolerances in the ANSI/IEEE NuBus specification.

There is room in the Macintosh Quadra 900 that could be used for a NuBus expansion card measuring 152.4 mm (6.0 inches) high and between 326.6 mm (12.858 inches) and 177.8 mm (7.0 inches) long. However, a card this size will fit in only the Macintosh Quadra 900 and is not guaranteed to fit in any future Macintosh computers. Foldout 5 shows this oversized card viewed from the component side. Like on the standard NuBus card, the NuBus connector is on the bottom edge of the card in the drawing, and the I/O connector is on the right side.

△ **Important** You should test all standard-size NuBus cards in the Macintosh modular platforms: the Macintosh IIcx, the Macintosh IIci, and the Macintosh Quadra 700. A standard-size NuBus card extends 326.6 mm (12.858 inches) and might interfere with the NMI and reset buttons in these machines. △

Card thickness must be 1.575 ±0.1906 mm (0.0062 ±0.0075 inch). Warpage must be controlled to within a 2.541 mm (0.10 inch) deviation from ideal.

Components may be placed anywhere within the unslashed area of the foldout drawing. The prohibited area along the top edge in the drawing applies to cards of any length. The five holes 3.38 mm (0.133 inch) in diameter are used only for Apple tooling purposes and are optional to you.

Components may not extend beyond the edge of the card in any direction. Component height must not be more than 15.246 mm (0.60 inch), measured from the card surface. No component or wire lead is allowed to extend more than 2.541 mm (0.10 inch) beyond the noncomponent side of the card.

The nominal spacing between centerlines of adjacent NuBus connectors is 22.869 mm (0.900 inch) in the Macintosh II, Macintosh IIx, Macintosh IIfx, and Macintosh Quadra 900 computers and 24.1395 mm (0.950 inch) in the Macintosh IIcx, Macintosh IIci, and Macintosh Quadra 700 computers.

The NuBus specification allows for an external connector plastics opening of only 74.55 mm by 11.90 mm. The Macintosh II and IIx allowed a significantly larger hole than the specification (80.00 mm by 17.00 mm). The external connector opening in the Macintosh IIcx is yet another size (75.61 by 14.00 mm), which is still larger than the NuBus specification. There is no guarantee, however, that future Macintosh computers will continue to have larger openings. Again, if you would like your cards to fit into more than one Macintosh computer, your design should adhere to the NuBus specification.

In the Macintosh IIcx, the intercard spacing is also different from that in other Macintosh computers with NuBus. Originally, in the Macintosh II and IIx, the intercard spacing was set to the minimum space (22.86 mm) allowed by the NuBus specification. In the Macintosh IIcx, this dimension was expanded to 24.13 mm. Because the intercard spacing is likely to continue to vary in future Macintosh computers, you should adhere to the NuBus specification to guarantee that your NuBus expansion card will fit in all Macintosh computers with the NuBus interface.

NuBus connector description

The NuBus connector on the card must be a 603-2-IEC-C096-M connector. See Chapter 5 for NuBus connector pin assignments. Figure 6-1 shows the version of that connector used on the Macintosh II Video Card. Figure 6-2 shows the NuBus mating connector on the main logic board of a Macintosh computer. Note that this is the same as the PDS connector used in the Macintosh SE and shown in Figure 17-6. For the Macintosh IIsi, the NuBus mating connector is not found on the main logic board. Instead, you must first install a NuBus adapter card, which is described in the section "Physical Implementation of the Macintosh IIsi NuBus Adapter Kit" later in this chapter. Once the adapter card has been installed, the NuBus mating connector will be present on the NuBus adapter card.

You can get Euro-DIN connectors meeting Apple specifications from

Amp Incorporated
Harrisburg, PA 17105

Because of high-volume production requirements, Apple purchases specially modified versions of the Euro-DIN connector from this vendor. However, you may purchase mating connectors of standard configuration from this or other vendors.

For EMI protection, a metal shield surrounds the I/O connector on the rear of the card. Appendix A, "EMI, Heat Dissipation, and Product Safety Guidelines," provides information on EMI reduction when a Macintosh computer with the NuBus interface is expanded. See Foldout 6 at the back of the book for a drawing of the I/O connector shield.

▲ **Warning** Foldout 6 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change. ▲

You can get the I/O connector shield meeting Apple specifications from

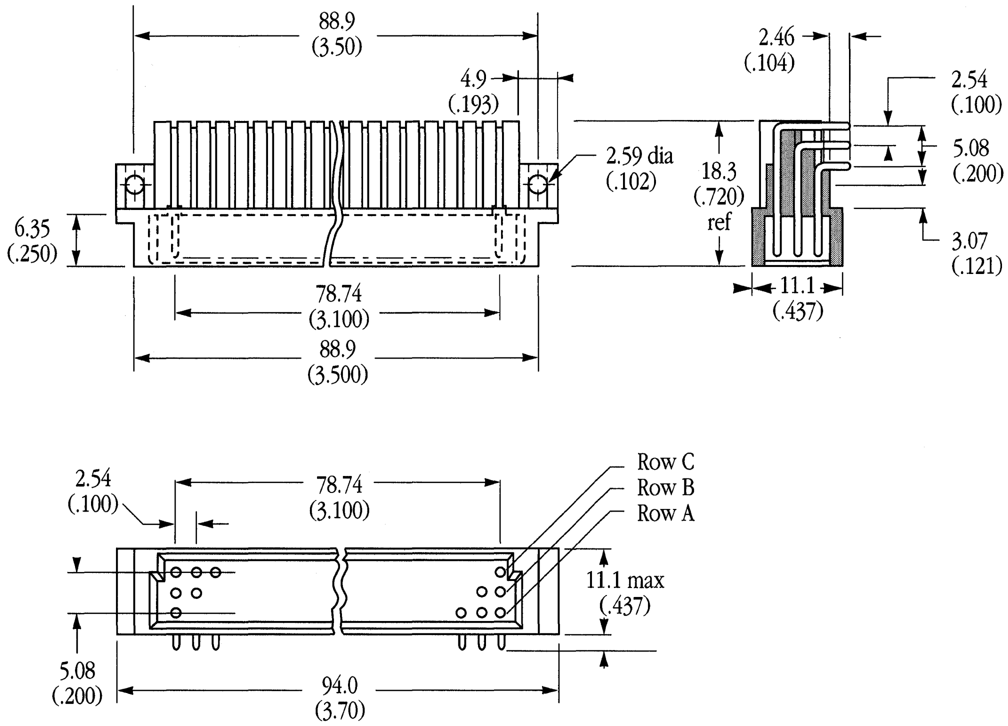
North American Tool and Die
San Leandro, CA 94577

The part number for the connector shield is 805-5101. Before ordering the connector shield, however, you must first obtain authorization from Apple Macintosh Developer Technical Support (MacDTS).

The type and number of I/O connectors (if required) are left to you, but they must meet dimensional constraints of the shield.

If auxiliary connectors are used, they must be no longer than 76.23 mm (3.0 inches). Please refer to the *Standard for a Simple 32-Bit Backplane Bus: NuBus,* ANSI/IEEE Std 1196-1990, for information about the location of auxiliary connectors.

■ **Figure 6-1** A 96-pin plug connector for a NuBus expansion card

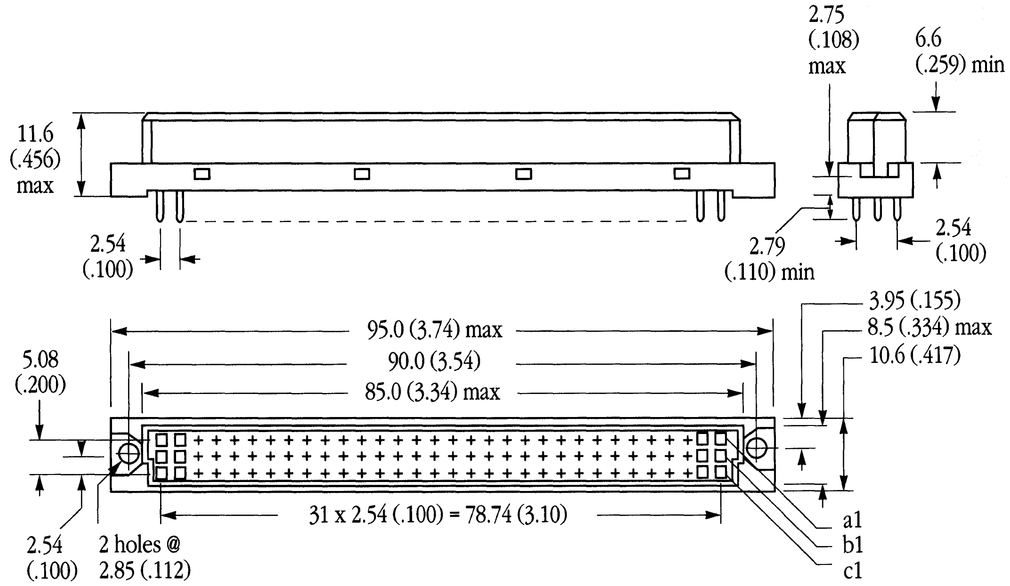


Three-row pin connector

96 contact positions
2.54 mm (.100 inch) spacing pins
Gold plated, 20 microinches, over nickel plate

Dimensions are in millimeters with inches in parentheses.

■ **Figure 6-2** A 96-pin socket connector on main logic board



Three-row socket connector
96 contact positions
2.54 mm (.100 inch) spacing sockets
Gold plated, 20 microinches, over nickel plate

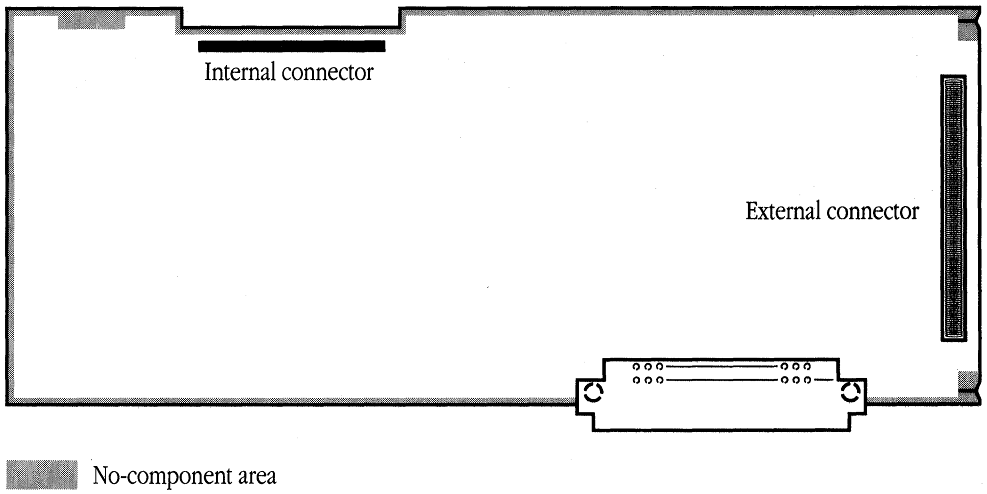
Dimensions are in millimeters with inches in parentheses.

NuBus expansion card internal connectors

Some NuBus card developers need to connect two expansion cards. The NuBus specification allows for this need with an auxiliary connector at the top of the card next to the no-component area. A ribbon cable must be used to connect the cards. The cable is run over the top of the card, and you must provide a slot in the card into which the cable fits. Figure 6-3 is an example of the correct way to implement your internal connector.

If you cut a slot at the top of your NuBus expansion card, you will have no problem with future Macintosh computers that provide NuBus expansion. The slot needs to be deep enough for the cable to be flush with the top of the card.

■ **Figure 6-3** Internal connector cabling slot for NuBus expansion card



The internal connector must have no parts that extend into the no-component area. If your connector has lock and eject tabs, similar to the internal SCSI connector, the tabs must be below the no-component area.

The no-component area is an area in which you may not put any component parts. The lid of a Macintosh computer has two fingers that hold the NuBus cards into place. These fingers are needed for stability, and they help to ensure that the cards will not be damaged in the event that the computer is jarred. If there are components in the no-component area, either the fingers will break the components or the lid will not fit correctly.

The no-component area is three-dimensional. As such, it covers the surface of the board as well as the distance next to the card (22.86 mm). You must not violate this space with either mounted parts or secondary logic boards.

Recommended heat dissipation guidelines

Apple recommends that NuBus expansion cards dissipate a maximum of 13.3 W of power. This total, which provides a comfortable margin for the major computer components, is arrived at as follows:

+5 V	@	2.0 A =	10.0 W
+12 V	@	0.175 A =	2.1 W
–12 V	@	0.1 A =	1.2 W
Total power =			13.3 W

Dissipation of more than 13.3 W of power by a card may cause excessive temperature rise on certain critical components. Apple studies indicate that at an ambient temperature of about 24°C, 13.3 W of dissipated power from the expansion card will cause an acceptable rise in average component case temperature to about 53°C. (Studies were conducted with an internal hard disk drive installed.) For additional information, refer to the section "Heat Dissipation Guidelines for NuBus Cards" in Appendix A.

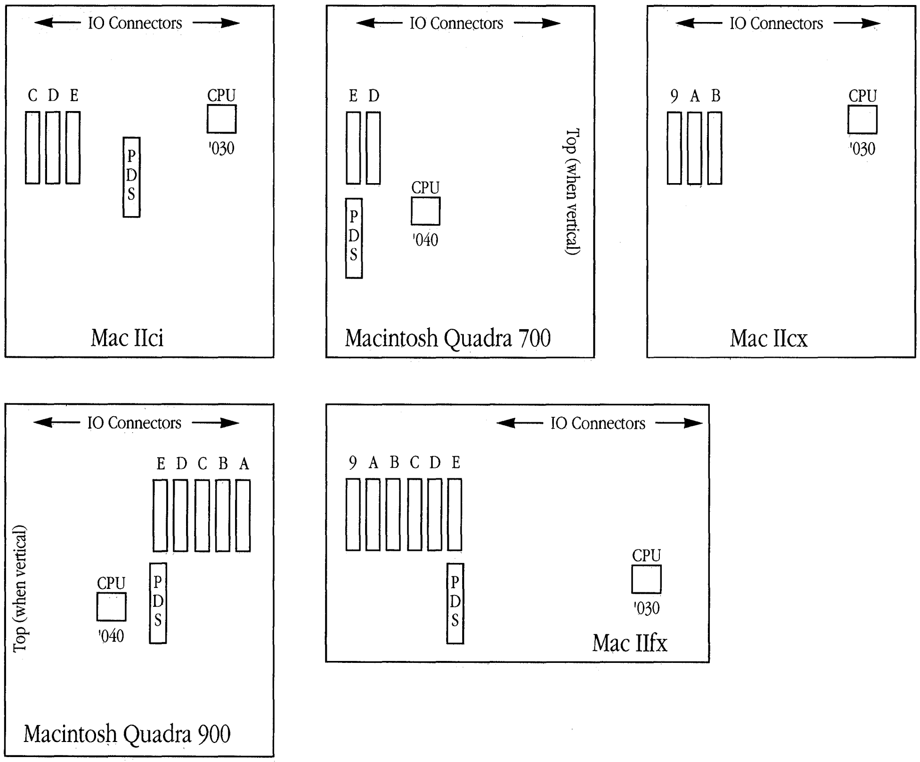
NuBus slot ordering

The list of Macintosh computers that offer the NuBus expansion interface is growing, and the variety of configurations is growing as well. There are Macintosh computers with one, two, three, five, and six available NuBus slots.

For most of the Macintosh computers, when the main logic board is viewed from above, NuBus slot ID ordering starts with a lower-number ID at the left side and increases from left to right. However, on the Macintosh Quadra 700 and the Macintosh Quadra 900, the NuBus slot ID ordering starts with higher ID numbers on the left and *decreases* from left to right. There should be no compatibility problems due to the physical ordering of NuBus slots on computers in the Macintosh Quadra family.

Figure 6-4 shows the slot ordering on the Macintosh Quadra 700 and the Macintosh Quadra 900, as compared with that on the Macintosh IIci, the Macintosh IIcx, and the Macintosh IIfx.

■ **Figure 6-4** NuBus slot ordering on Macintosh IIci, Macintosh Quadra 700, Macintosh IIcx, Macintosh Quadra 900, and Macintosh IIfx computers



Physical implementation of the Macintosh IIsi NuBus adapter kit

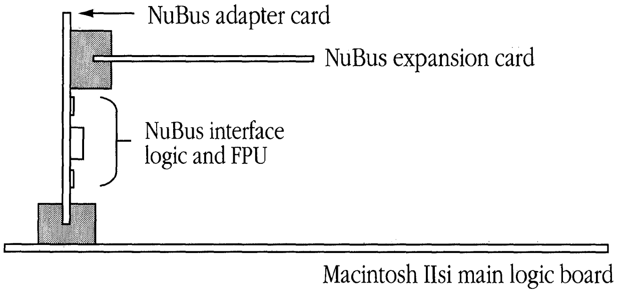
A NuBus adapter kit, available from an authorized Apple dealer, allows a customer to install a NuBus card in the Macintosh IIsi computer and have it function exactly as if it were in any other Macintosh computer with the NuBus interface. The NuBus adapter card includes two different connectors. One is a 120-pin plug connector that mates with the Euro-DIN 120-pin socket connector on the left side (viewed from the front) of the main logic board. The adapter card mounts vertically in this connector. The other is a 96-pin socket connector (the same as the NuBus connectors on the main logic boards of other Macintosh computers). The NuBus card plugs into this connector and is positioned horizontally over the main logic board.

The NuBus vertical adapter card acts as a translator between the 68030 signals on the main logic board and the signals on the NuBus expansion card. It contains the same custom NuChip 30 that is found in the Macintosh IIci, other electrical components that make up the NuBus interface logic, and a 20 MHz MC68882 FPU. All the translation is transparent, so that NuBus cards will work successfully in the Macintosh IIsi.

The adapter card also includes a bracket that adapts the NuBus card's connector to the opening in the back of the Macintosh IIsi computer. This bracket provides both EMI protection and support for the card. Two screws are included in the adapter card kit to secure the bracket to the opening in the chassis. The top cover of the Macintosh IIsi computer includes grooves that hold the NuBus card in place when the cover is closed. There is just enough gap between the top cover and bottom half of the case for a 1.575 mm thick card, which is Apple's specification for a NuBus card. Any size NuBus card specified by Apple can be accommodated in the Macintosh IIsi computer.

Figure 6-5 is a sketch showing a NuBus card and its adapter card installed on the main logic board of a Macintosh IIsi computer. Notice that, unlike the 68030 Direct Slot adapter card specified in Chapter 15, the NuBus card has other electrical components in addition to the FPU, including the custom NuChip 30 and associated transceiver chips that make up the NuBus interface logic. Although functionally identical to that of the Macintosh IIsi, the NuBus interface logic in other Macintosh computers is on the main logic board.

- **Figure 6-5** Installing a NuBus card and adapter on the Macintosh IIsi main logic board


NuBus adapter card
NuBus expansion card
NuBus interface logic and FPU
Macintosh IIsi main logic board

Chapter 7 **NuBus Card Memory Access**

This chapter describes how cards connected to a Macintosh computer through the NuBus slots can access address space. The discussion is in three sections:

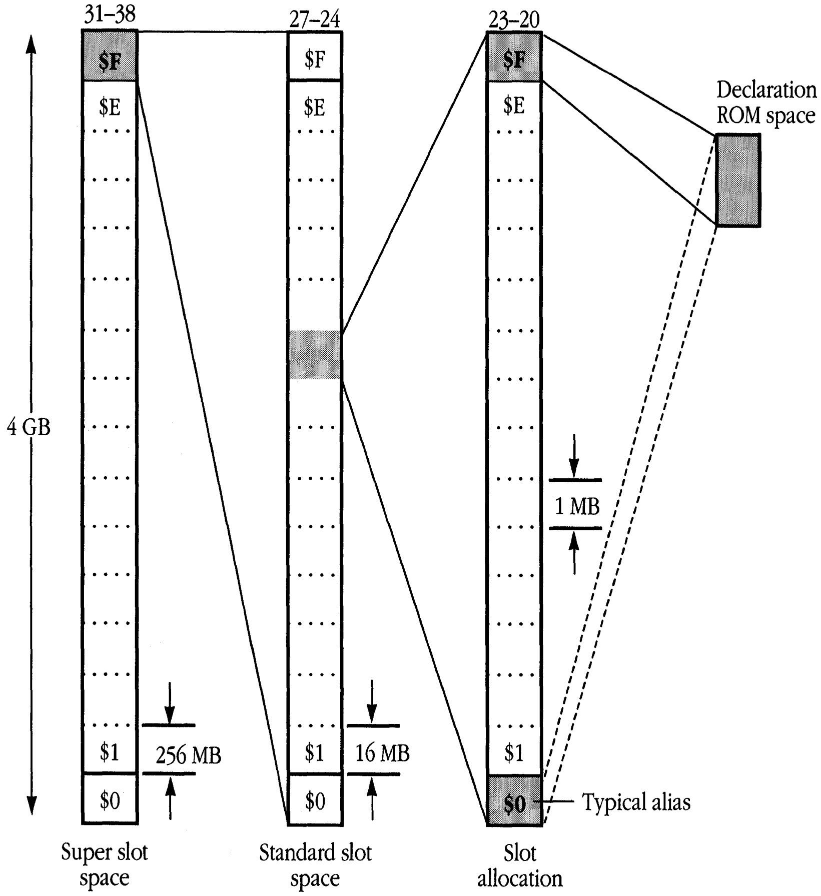
- a general description of the NuBus address space and how it is accessed in both 24-bit and 32-bit modes
- a discussion of how the NuBus address space is allocated, including its mapping to the address space in a Macintosh computer
- a description of the bit structure of NuBus messages and how it differs from the microprocessor bus architecture

In addition to the memory areas it uses for its own operations, every NuBus card must contain a declaration ROM area. The declaration ROM contains certain standard data structures that are used by the Macintosh Slot Manager. These data structures are defined in Chapter 8, "NuBus Card Firmware."

NuBus address space

The NuBus architecture allows full 32-bit addresses, providing 4 GB of address space. The upper one-sixteenth (256 MB) of the NuBus address space is called the standard slot space. As shown in Figure 7-1, this addressing region is further divided into 16 regions of 16 MB apiece, each of which constitutes the standard slot space for one possible slot ID. NuBus addresses of the form $Fsxx xxxx (that is, $Fs00 0000 through $FsFF FFFF) address the standard slot space that belongs to the card in slot s, where *s* is an ID digit in the range $9 through $E. Because Macintosh computers with NuBus use only slot IDs $9 through $E, only the six standard slot spaces $F9xx xxxx through $FExx xxxx are actually used.

■ **Figure 7-1** NuBus address space



For convenience, this section refers to a NuBus configuration of six slots represented by slot IDs $9 through $E. However, not all Macintosh computers have six NuBus expansion slots. Table 7-1 lists the slot IDs available for each Macintosh computer with the NuBus interface, and their corresponding slot spaces.

■ **Table 7-1** NuBus slot IDs and slot spaces for Macintosh computers

Macintosh model	NuBus slot IDs	NuBus slot spaces
Macintosh IIcx	$9, $A, $B	$F9xx xxxx–$FBxx xxxx
Macintosh IIci	$C, $D, $E	$FCxx xxxx–$FExx xxxx
Macintosh IIsi†	$9	$F9xx xxxx
Macintosh Quadra 700	$D, $E	$FDxx xxxx–$FExx xxxx
Macintosh Quadra 900	$A, $B, $C, $D, $E	$FAxx xxxx–$FExx xxxx
Macintosh II, IIx, and IIfx	$9, $A, $B, $C, $D, $E	$F9xx xxxx–$FExx xxxx

† A NuBus adapter card must be installed in the Macintosh IIsi before a NuBus expansion card is installed.

This system of fixed address allocations, based solely on a card's slot location, makes it possible for you to design cards that are free of jumpers and configuration switches.

▲ **Warning** Whenever possible, use 32-bit addressing conventions and methods. This will be your best guarantee of future software compatibility. ▲

When a NuBus card needs to address more than 16 MB, it can access an additional region of the NuBus address space. The area from $9000 0000 through $EFFF FFFF is called super slot space. It is divided into regions of 256 MB each. NuBus addresses of the form $sxxx xxxx (that is, $s000 0000 through $sFFF FFFF) address the super slot space that belongs to the card in slot s.

Figure 7-1 also shows the card's declaration ROM space, discussed in Chapter 8.

As explained in *Inside Macintosh,* a Macintosh computer operates in either 32-bit or 24-bit mode. In 32-bit mode, it can access all the address space in both the standard slot and super slot spaces of any slot card. In 24-bit mode, it can address only 1 MB of each card's standard slot space. This first megabyte of standard slot space is called **minor slot space.** In 24-bit mode, the computer hardware translates 24-bit addresses of the form $sx xxxx into 32-bit addresses of the form $Fs0x xxxx, where *s* is a digit in the range $9 through $E.

Addresses of the form $Fssx xxxx access the same NuBus slot in both 24-bit and 32-bit modes. However, if you use an address of this form in 24-bit mode, the system will translate it into a NuBus address of $Fs0x xxxx. In 32-bit mode it will remain unchanged. Hence, if you need less than 1 MB of address space to be accessible from NuBus, you should design your card to use only bits /AD19–/AD0. By ignoring bits /AD23–/AD20, you guarantee that addresses of the form $Fssx xxxx will be valid in both 24-bit and 32-bit modes.

The computer hardware translates other 24-bit addresses above $7F FFFF into different 32-bit addresses. The full translation algorithm is shown in Table 7-2.

■ **Table 7-2** 24-to-32-bit address translations

24-bit address range	32-bit address range	Notes
$00 0000–$7F FFFF	$0000 0000–$007F FFFF	
$80 0000–$8F FFFF	$4000 0000–$400F FFFF	
$s0 0000–$sF FFFF	$Fs00 0000–$Fs0F FFFF	*s* in range $9 through $E
$F0 0000–$FF FFFF	$5000 0000–$500F FFFF	

Address allocations for Macintosh computers with NuBus

All of the existing address space is accessible from NuBus. It is mapped onto the NuBus address space as shown in Table 7-3.

When the microprocessor accesses 32-bit addresses in the range $6000 0000 through $FFFF FFFF (except for $6000 0000 through $7FFF FFFF in the Macintosh IIfx and $F0xx xxxx in all other Macintosh computers), it initiates a NuBus transaction. The mapping shown in Table 7-3 is correct for current members of the Macintosh family. Future Macintosh products may have different mappings.

■ **Table 7-3** NuBus address mapping

24-bit addresses	32-bit addresses		
from processor	from processor	NuBus address	Used to access computer
$xx00 0000 to $xx7F FFFF	$0000 0000 to $007F FFFF	$0000 0000 to $007F FFFF	Present RAM.
	$0080 0000 to $3FFF FFFF	$0080 0000 to $3FFF FFFF	RAM expansion.
$xx80 0000 to $xx8F FFFF	$4000 0000 to $4FFF FFFF	$F080 0000 to $F0FF FFFF	ROM (aliased).
$xxF0 0000 to $xxFF FFFF	$5000 0000 to $5FFF FFFF	$F000 000 to $F070 FFFF	I/O (aliased). Do not access from a slot card.
	$6000 0000 to $6FFF FFFF	$6000 0000 to $6FFF FFFF	Slow PDS slot space for Macintosh IIfx (presently unused for other CPUs).
	$7000 0000 to $7FFF FFFF	$7000 0000 to $7FFF FFFF	Fast PDS slot space for Macintosh IIfx (presently unused on other CPUs).
	$8000 0000 to $8FFF FFFF	$8000 0000 to $8FFF FFFF	Presently unused.
	$9000 0000 to $EFFF FFFF	$9000 0000 to $EFFF FFFF	Super slot space, slots $9 to $E. On Macintosh Quadra 700 and Macintosh Quadra 900, $9 is used for video slot space; on Macintosh IIsi, $E is used for video slot space.
$xxF0 0000 to $xxFF FFFF	$F000 0000 to $F0FF FFFF[†]	$F000 0000 to $F0FF FFFF	Slot $0 (Macintosh system).[†]
	$F100 0000 to $F8FF FFFF	$F100 0000 to $F8FF FFFF	These addresses are unaccessible.
$xxs0 0000 to $xxsF FFFF	$Fs00 0000 to $Fs0F FFFF or $Fs10 0000 to $FsFF FFFF	$Fs00 0000 to $Fs0F FFFF or $Fs10 0000 to $FsFF FFFF	Standard slot space, slot s (*s* in the range $9–$E). On Macintosh Quadra 700 and Macintosh Quadra 900, $9 is used for video slot space; on Macintosh IIsi, $E is used for on-board video.
	$FF00 0000 to $FFFF FFFF	$FF00 0000 to $FFFF FFFF	Presently unused.

[†] If the microprocessor attempts to access addresses in this range, it will immediately generate a bus error (/BERR) exception. No NuBus transaction will take place.

Slot allocations

In 24-bit mode, the lower 1 MB of each card's standard slot space is mapped onto a part of the 24-bit Macintosh address space. This address space is used for communication between the card in that slot and the computer. For example, NuBus addresses $F900 0000 through $F90F FFFF correspond to 24-bit Macintosh addresses $90 0000 through $9F FFFF and are used by slot $9. All the rest of each slot's NuBus address allocation is available for other uses by the card in that slot and may also be addressed by Macintosh computers in 32-bit mode and by cards in other slots. These allocations are listed in Table 7-4.

■ **Table 7-4** Slot allocations

Slot	24-bit addresses	NuBus super slot space	NuBus standard slot space
$9	$90 0000–$9F FFFF	$9000 0000–$9FFF FFFF	$F900 0000–$F9FF FFFF
$A	$A0 0000–$AF FFFF	$A000 0000–$AFFF FFFF	$FA00 0000–$FAFF FFFF
$B	$B0 0000–$BF FFFF	$B000 0000–$BFFF FFFF	$FB00 0000–$FBFF FFFF
$C	$C0 0000–$CF FFFF	$C000 0000–$CFFF FFFF	$FC00 0000–$FCFF FFFF
$D	$D0 0000–$DF FFFF	$D000 0000–$DFFF FFFF	$FD00 0000–$FDFF FFFF
$E	$E0 0000–$EF FFFF	$E000 0000–$EFFF FFFF	$FE00 0000–$FEFF FFFF

Slot $0 corresponds to the Macintosh computer itself. It addresses the 16 MB of NuBus slot space from $F000 0000 through $F0FF FFFF. The microprocessor cannot access slot $0.

NuBus bit and byte structure

The NuBus bit structure is not the same as the bit structure of the MC68020 bus, the MC68030 bus, or the MC68040 bus. To achieve byte-addressing consistency, the Macintosh computers perform byte swapping of data between the microprocessor and the NuBus. This section explains the rationale and details of this implementation.

Unfortunately, there is no universal agreement about what the significance of a given addressed byte should be within a larger unit. For example, within a NuBus word, byte 3 is the most significant byte; in the 68020, the 68030, and the 68040 microprocessors, byte 3 is the least significant byte of a longword (32-bit) value.

In designing the Macintosh family of computers, a choice had to be made about whether to preserve the significance of bytes between the NuBus and the processor or to preserve byte-addressing consistency. Note that this choice deals with how the 4 bytes within a NuBus word and a processor longword are connected to each other.

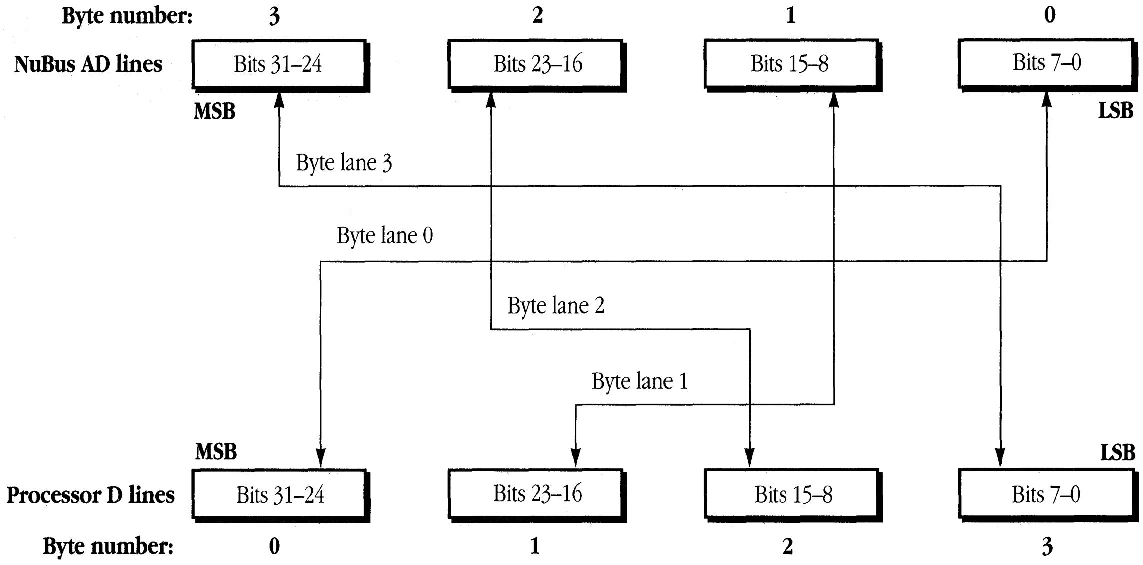
Apple chose to preserve byte-address consistency; each of the 4 bytes of the processor is connected to its corresponding NuBus **byte lane.** A byte lane is the route by which bytes are transferred between the NuBus and the computer's microprocessor. That is, byte *n* of the processor is connected to NuBus byte lane *n,* as shown in Figure 7-2. Byte lanes are numbered to reflect the bytes they carry: that is, byte lane 0 carries byte 0, byte lane 1 carries byte 1, and so on. NuBus encodes the least significant bits in its data word into byte 0 and the most significant bits into byte 3. The microprocessor does the reverse: it places its least significant bits in byte 3 and its most significant bits in byte 0. Byte-lane routing is performed automatically by Macintosh computers. Only the bytes are swapped, not bits within bytes. Notice in Figure 7-2 that bit numbers do not have a direct correspondence between the processor and the NuBus. For example, bits D31–D24 (byte 0) of the processor are connected to bits AD7–AD0 (byte lane 0) of the NuBus.

The significance of a byte within a larger item is reversed in this process. That is, the most significant bit of a NuBus word is in byte lane 3, while the most significant bit of a 68020, 68030, or 68040 longword is in byte 0. Thus, there is an apparent swapping of the bytes between the world of the microprocessor and NuBus; this is referred to as **byte swapping.**

For many cards, byte swapping is not important. However, for cards that communicate with processors of different byte ordering, very careful attention must be paid to the NuBus interface. An Intel 80386 microprocessor, for example, has byte ordering identical to that of NuBus; that is, the least significant bit of an 80386 word is byte 0, and the most significant bit is byte 3.

Transferring data by *bytes* between such a processor and the NuBus always produces the correct value. However, if the MC68020, MC68030, or MC68040 reads a NuBus *word* from an 80386 on a card, it reads a value whose bytes are swapped in significance. For example, a word read of a location within the 80386 card that contains a 32-bit value of $1234 5678 is seen as $7856 3412 by the Macintosh processor because of the byte swapping.

■ **Figure 7-2** Byte-lane mapping



Communication between a Macintosh computer and a NuBus card may use any combination of one or more byte lanes. This subject is discussed in more detail in the section "The Format Block" in Chapter 8.

Although cards may communicate with each other over the NuBus in any format, all communication with the computer (including communication between a card's declaration ROM and the Macintosh Slot Manager) must conform to the microprocessor bus format. This may require byte swapping when word and long data types are used.

Byte smearing

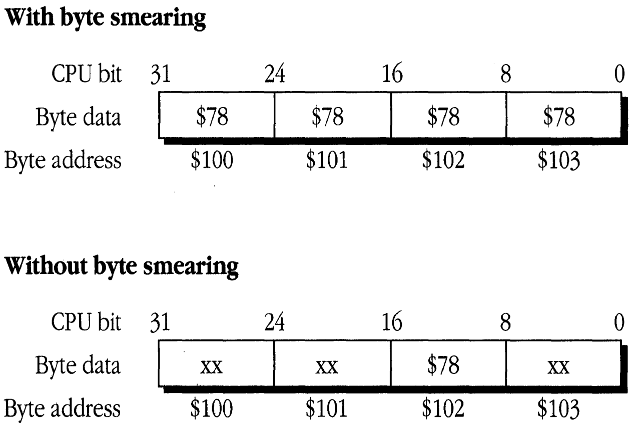
The MC68020 and MC68030 processors share a characteristic that causes the data for byte and word transfers to be duplicated, or smeared, across all 32 data lines. This feature is called **byte smearing.** For more information about byte smearing, refer to the *MC68020 16/32-Bit Microprocessor User's Manual* and the *MC68030 32-Bit Microprocessor User's Manual.*

For an example of byte smearing, suppose your code includes the following instructions. The instructions write 4 bytes of data into a 32-bit register, D0, and then attempt to write a byte of that data into a memory location.

```
MOVE.L    #$12345678,D0       ; Insert data into D0
MOVE.B    D0,$102             ; Write a single byte of data from D0
```

The data actually placed on the data bus, with and without byte smearing, is shown in Figure 7-3.

■ **Figure 7-3** Effect of byte smearing



With byte smearing

Without byte smearing

With byte smearing, the byte of data is replicated across all the byte lanes. Without smearing, the other bytes are undefined. A similar replication of data can occur with word transfers.

△ **Important** The byte-smearing feature does not exist on 68040-based machines. If you have software or hardware that depends upon byte smearing, you must revise it. Likewise, you should be aware of this limitation if you are developing new hardware or software. △

Chapter 8 **NuBus Card Firmware**

This chapter describes the Slot Manager and the firmware that must be included on cards that communicate with a Macintosh computer through the NuBus protocol. Such firmware is normally in a ROM area on the card called the **declaration ROM** (also known as the **configuration ROM**).

The discussion in this chapter is divided into the following parts:

- an introduction to the Slot Manager and the card's declaration ROM firmware
- a list of data types used by the Slot Manager and the declaration ROM firmware
- a description of the required internal structure of the declaration ROM firmware
- a description of the additional internal data structures that are unique to the declaration ROM of a video card
- a description of the Macintosh Coprocessor Platform and the A/ROSE operating system, designed to run on intelligent NuBus cards

Sample code for a typical NuBus card has been included in Appendix B. This sample code includes a sample of declaration ROM code, primary initialization code, and secondary initialization code.

An introduction to the firmware

This section examines the relationship between the Slot Manager and the declaration ROM, and explains some major concepts that you must understand before you can implement the declaration ROM firmware. It is important that you read and comprehend the information in this overview; if you understand the relationship of the Slot Manager to the declaration ROM, and the concept of slot resources, you will find it much easier to grasp the detailed information contained in the rest of this chapter.

◆ *Note:* In other chapters of this book, a distinction is made between *board* (a printed-circuit board that is a permanent part of the computer) and *card* (a printed-circuit board that can be inserted and removed). However, because the firmware terminology uses the term *board* to mean a removable card, that distinction is not made in this chapter—both *card* and *board* mean a removable card.

The Slot Manager and the declaration ROM

The Slot Manager and declaration ROM have three main goals:

- to provide a standard mechanism for recognizing the presence of an expansion card in the computer
- to describe data structures and provide a programmable interface that simplifies access to information about the card's functionality, as well as to necessary data and code to support each device
- to allow you to insert an expansion card into any slot without configuring switches or installing special software

If a valid declaration ROM is present, then system software, applications, and drivers can take advantage of the Slot Manager's library of routines to accomplish these goals.

The **Slot Manager** is a group of routines that communicate with the declaration ROM firmware on an expansion card. It is located in the ROM of a Macintosh computer. The Slot Manager originally only worked with NuBus expansion cards; but by using pseudoslot design, the Slot Manager can also be used with processor-direct slot (PDS) expansion cards. Pseudoslot design and PDS expansion cards are discussed in Part II of this book.

The Slot Manager gets information from the declaration ROM and provides it to the application program to identify an expansion card in a NuBus slot, to define the functions that card can perform or to pass it other data. It does this by determining what, if any, expansion cards are in the Macintosh computer at startup time and by fetching identification, functional, and other information from the firmware of each card identified. The Slot Manager places the information it gathers into data structures that your application can access by using the Slot Manager routines. Slot Manager information in *Inside Macintosh* describes these routines in detail.

The declaration ROM is an area on a NuBus expansion card that contains firmware that identifies the card and its functions and that allows the card to communicate with the computer through the Slot Manager routines. However, communication with the Slot Manager is possible only if you configure your card's declaration ROM firmware properly. The declaration ROM provides all the necessary data for you to install an expansion card in a slot and ensure that it will work without setting any DIP switches or loading any special software.

Your card's declaration ROM firmware can be implemented in any of three physical widths: 8, 16, or 32 bits. It must include these elements:

- a format block
- an sResource directory
- an sResource (slot resource) for each function on the card plus one unique sResource called a *board sResource*

The section "Firmware Structure," later in this chapter, defines each of these elements and describes their firmware structure in detail. All of the elements are important; but before you try to implement your firmware, you should become thoroughly familiar with two major concepts: the *sResource* and the `sRsrcType` entry of an sResource.

sResources

The combination of the Slot Manager and declaration ROM identifies your expansion card and allows the computer, and high-level applications and drivers, to communicate with it. The way they do this is through one or more **sResources (slot resources).** The small *s* indicates a slot resource as opposed to a real Macintosh resource. Don't confuse sResources on expansion cards with standard Macintosh resources; they are different, although related conceptually. The firmware in your card's declaration ROM defines these sResources.

While most sResources define a function or capability of the expansion card, some sResources may contain only data—for example code, icons, special fonts, or vendor-defined data.

There is typically one sResource for each function a card can perform plus one (and only one) unique sResource called a **board sResource.** An sResource relating to a specific function a card can perform is called a **functional sResource.** It provides information about that particular function to high-level applications that are interested in getting access to the function. Functional resources are usually associated with a driver and are used to point to the driver.

Most cards perform only one function. For example, a modem card might perform only a modem function, a video card a video function, and so on—each of these cards would have only one functional sResource. However, it is possible to build an expansion card with many functions. An example is a multifunction card that contains a parallel port, a serial port, and a modem. In this case, the card's declaration ROM could have three functional sResources—one for each function—as well as the required board sResource.

You need a functional sResource for each of a card's functions so that higher-level software (applications looking for the function or drivers wanting to communicate with the card) can query the Slot Manager, which will find and return the location of the card. This allows applications to use compatible cards (either later versions of cards made by the same manufacturer or compatible cards made by other manufacturers), resulting in a larger installed base for the application without having to make a revision each time a new card capable of handling the particular function becomes available.

Although functional sResources are desirable and beneficial, you are not required to include them in the declaration ROM; the board sResource, however, is always required. If the board sResource is absent or invalid, the Slot Manager marks the slot where the card is located as invalid, and Slot Manager calls to that slot do not work.

The board sResource is, in a sense, a special case of a functional sResource. It provides a handy place to store card-related data that identifies the expansion card. This data includes entries such as the primary initialization routine that is called at system startup time, the board name, vendor identification, and anything else that you may want to identify.

How sResources are implemented

All sResources are implemented as lists of sResource entries terminated by a special `EndOfList` element. Each entry is a 32-bit record that consists of an 8-bit ID field and a 24-bit field that consists of either data or a signed offset to another structure. To get information from a declaration ROM, use the Slot Manager to find the beginning of the sResource, searching by either the ID number or its type entry (see the next section). Within each sResource list, the Slot Manager finds entries by searching in ascending order for an ID that you specify.

An sResource has several entries, some that are required and others that are optional, depending on your needs. For example, an sResource always needs an sRsrcType entry and an sRsrcName entry to identify it. If the sResource is device oriented, other entries you might include are sRsrcIcon, sRsrcDrvrDir, sRsrcLoadRec, and so on. The section "Apple-Defined sResource Entries," later in this chapter, describes all of these sResource entries in detail. But for now, it's important that you focus your attention on the sRsrcType entry. If you do not have correct information in the fields of this entry, applications and drivers cannot recognize the function provided or the special features of your card; the Slot Manager may even mark the slot as being empty.

The sRsrcType entry

Every sResource must include an sRsrcType entry whose fields identify that particular sResource. When applications and drivers communicate (via the Slot Manager) with your card's declaration ROM, they use the information in the fields of the sRsrcType entry of each sResource to identify the functions the card performs (in the case of functional sResources) or to identify the card itself (in the case of the board sResource).

The format of an sRsrcType entry consists of two 32-bit-long integers divided into four major fields that are hierarchical in structure as shown here.

```
Category (bits 30–16; bit 31 is reserved for Apple's use)
    cType (bits 15–0)
        DrSW (16–31)
            DrHW (15–0)
```

Following is a brief description of each of the `sRsrcType` fields in a typical functional sResource.

`Category` The `Category` field identifies a unique broad functional category such as `Display`, `Network`, or `Memory`. (There are many predefined categories; these are just a few of them. Some cards can use predefined categories, but others will need new categories defined for them.)

`cType` The `cType` field narrows down the `Category` field. Under a given `Category`, you use the `cType` field to identify a subtype of that `Category`. For example, under `Category Display` there might be `cType` entries such as `Video` and `LCD`. (For commonly predefined `Category` types, there are often predefined `cType` entries.)

`DrSW` Continuing down the hierarchy, there are `DrSW` fields that identify the driver software interfaces that apply to a given `Category` and `cType`. The `DrSW` field identifies a driver for a particular resource. For example, under `Category Display` and `cType Video` a typical predefined driver software interface would be one defined by Apple to work with QuickDraw using the Macintosh Operating System frame buffers. Note that at this point in the hierarchy, a function's architecture has been defined down to the software interface level.

`DrHW` Finally, under the `DrSW` field you use the `DrHW` field to identify a specific hardware device, for example, the Apple Macintosh II Video Card.

◆ *Note:* Apple Macintosh Developer Technical Support (MacDTS) assigns alphanumeric strings and hexadecimal values that define the fields in the `sRsrcType` entry of each of your sResources. Instructions for obtaining these values are provided later in this chapter in the section "Obtaining Card Identification and `sRsrcType` Values From MacDTS."

MacDTS can assign new definitions and values to the `sRsrcType` fields of a card's functional sResources if no predefined categories or types exist. For example, suppose that the Widget company is developing a fractal card. Since there are no predefined `sRsrcType` fields for a fractal card's functional sResource, MacDTS might assign definitions such as `CatComputational`, `TypFractal`, `DrSWWidget`, and `DrHWWidget`, along with corresponding hexadecimal values.

While the previous discussion shows that the values assigned to the `sRsrcType` fields of functional sResources can vary for each function, it is important to remember that the values assigned to the `sRsrcType` fields of a board sResource are fixed and cannot change.

How to configure the sRsrcType fields for video card sResources

Because a QuickDraw-compatible video card is one of the most common expansion cards, this section uses the Macintosh II Video Card as an example to explain how to configure the `sRsrcType` fields for a video card's sResources. The Macintosh II Video Card satisfies the requirements of QuickDraw and the Macintosh Operating System but does not perform any unique functions that only a special-purpose application could benefit from.

◆ *Note:* In Chapters 8 through 11, reference is made to QuickDraw, Color QuickDraw, and 32-bit QuickDraw. Color QuickDraw is an extension of the original QuickDraw; it uses indexed color. After Color QuickDraw was made available, 32-bit QuickDraw was introduced as a system extension (an `'INIT'` resource) to handle direct color. (The difference between indexed and direct color is discussed in the section "Identifying Direct Devices" later in this chapter.) In System 7, however, the direct color features of 32-bit QuickDraw have been included in Color QuickDraw, thus eliminating the need for a separate system extension.

Assume that you are designing a video card. Since the card probably performs only one function, its declaration ROM should include one functional sResource to declare the video function plus the required board sResource to identify the card. The four `sRsrcType` fields of the video card's functional sResource and board sResource are explained and illustrated in the following sections.

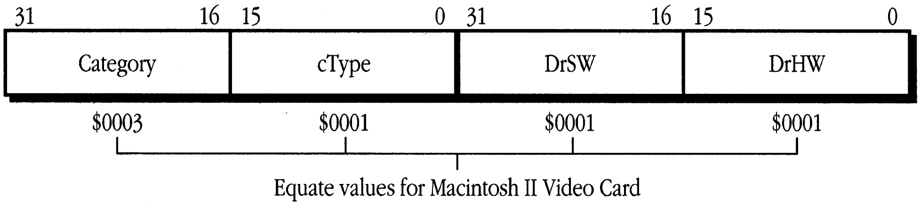
sRsrcType fields for a video card functional sResource

Figure 8-1 illustrates the format and hierarchical structure of the fields in the `sRsrcType` entry of a functional sResource that defines the display function of a QuickDraw-compatible video card. This example uses the values assigned to the Macintosh II Video Card, often referred to as the *TFB* card. The combined value of the `sRsrcType` fields for the Macintosh II Video Card is $0003 0001 0001 0001. The hierarchical structure is illustrated by the indentations of the fields.

Notice that all of the hardware devices identified in Figure 8-1, starting with `DrHWTFB`, adhere to the `DrSWApple` software interface, since they are nested under `DrSWApple`. Thus, if a company decides to make a new video card that adheres to the Apple driver software interface, MacDTS needs to assign it only a new `DrHW` value, for example, `DrHWproductE`, to differentiate it from other video cards.

For more in-depth information on the video driver software interface, refer to the section "Video Driver Routines" in Chapter 9.

- **Figure 8-1** Example of `sRsrcType` fields for a functional sResource

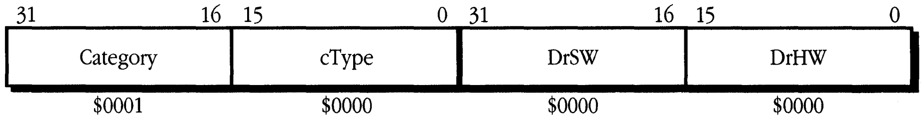


CatDisplay	Equate value of $0003
TypVideo	Equate value of $0001
DrSWApple	Equate value of $0001
DrHWTFB	Equate value of $0001
DrHWproductA	Equate value of $0002
DrHWproductB	Equate value of $0003
DrHWproductC	Equate value of $0004
DrHWproductD	Equate value of $0005
DrSWcompanyA	Equate value of $0002
TypLCD	Equate value of $0002

sRsrcType fields for a video card board sResource

Remember that a board sResource is nothing more than a unique type of sResource that identifies the card rather than a function performed by the card. The board sResource includes `sRsrcType` fields that identify its function as that of being an expansion card. The format of these fields and their hierarchical structure are shown in Figure 8-2. Note that the values assigned to the `sRsrcType` fields of a board sResource are always the same. For example, `Category` is always $0001, and `cType`, `DrSW`, and `DrHW` are always $0000.

■ **Figure 8-2** Example of `sRsrcType` fields for a board sResource



CatBoard	Equate value of $0001
TypBoard	Equate value of $0000
DrSWBoard	Equate value of $0000
DrHWBoard	Equate value of $0000

How QuickDraw interacts with the Slot Manager and declaration ROM

This section describes how QuickDraw and the Slot Manager use the sResource information in the video card's declaration ROM.

When the system first comes up, the Start Manager makes a call to the Slot Manager requesting all expansion cards whose `sRsrcType` fields include `CatDisplay`, `TypVideo`, and `DrSWApple`. The QuickDraw software needs the information in these fields from all QuickDraw-compatible video cards. (The `DrHW` field is masked to indicate that QuickDraw does not care about this field and assumes that the card's video driver will handle the specifics of the hardware.) QuickDraw is concerned only with the video devices that conform to the Apple driver software interface. QuickDraw cannot support incompatible driver software interfaces. If it tried, it might make an incorrect control call, or use a wrong parameter, and so on. Thus, if you configure your `DrSW` field for a proprietary driver software interface that is not Apple compatible, your card will not work with QuickDraw

When the Slot Manager gets the request, it finds each video card that meets the above `sRsrcType` requirements. For example, you could have a different manufacturer's video card in each slot of a Macintosh II, and the Slot Manager would find all of them, because Apple has predefined the `DrSWApple` field, and all of the existing video cards are compatible with this driver software interface.

Apple manufactures the Macintosh II Video Card, whose DrHW field is identified as DrHWTFB, but Apple (or any company) could make any number of new video cards and each would have a different DrHW identification; since the driver software interface does not change, QuickDraw would not be affected at all. This gives you the advantage of not having to revise the software every time a new card becomes available as long as you adhere to the predefined Apple driver software interface.

You can also apply the preceding concept to any manufacturer who wishes to define their own Category, cType, and DrSW interface. When they do this, many different applications can work with a given card or cards (for example, various communication applications can run on one or more modem cards), or one application can work with a variety of cards (as shown in our previous discussion of QuickDraw). As an example, a customer could buy a new modem card, and instead of having to purchase a new application, their existing application would still work without having to be upgraded. Following this concept allows flexibility in your designs and broadens the market for your cards and applications.

Sometimes the DrHW field is very important and should not be masked as "don't care." For example, assume that a company found a bug in their expansion card's ROM and came out with a software patch for it. The patching software would have to locate the card so that it could apply the patch. To do this, the patching software would ask the Slot Manager to find all cards whose sRsrcType fields match down to and including the DrHW level. Without the DrHW field, the Slot Manager could not differentiate between two different manufacturers' cards and would not know which ROM to patch.

Summary of firmware design objectives

Expansion cards for the Macintosh require a declaration ROM if they are to be recognized by the system. This ROM contains both information that describes the capabilities of the card and executable code that is customized for each card. Because all card-specific code can be in the declaration ROM, a properly designed declaration ROM can often be used upon installation, with no additional software configuration required. If the size of the declaration ROM is limited, the code sections can be executed and their equivalents loaded from disk when needed. Note that video and hard disk cards require that the device driver be resident in the declaration ROM, since the startup devices in each of these categories are opened before the file system is available. Packaging all device-specific code in the declaration ROM greatly simplifies the use of an expansion card.

The declaration ROM should be completely self-contained. But if necessary, most executable sections can be easily and effectively overriden by an initialization file at startup time.

Properly designed applications should rarely need to search for a specific hardware variant (DrHW). Cards that have the driver in ROM (or separate from the application) allow the application program to mask off the DrHW field so that the application can work with different versions of the card. This provides the advantage of less maintenance for the application programmer, and a higher customer satisfaction level, since the application will still run even if the customer buys a later version of your card.

Well-designed applications adhere to the driver specification and should have no direct dependencies on the specific hardware implementation. By following the hierarchical structure in the design of your card's declaration ROM, you ensure maximum compatibility among the various products available, while allowing the user a great deal of flexibility in configuring the system.

If you publicly define a driver software interface for your card (as Apple has done with the video driver), then other manufacturers can develop applications that use your card because they know your driver can support the underlying hardware no matter what it is. Also, other manufacturers can make expansion cards that conform to the driver software interface, and applications (including one you may write) should be able to work with them.

Obtaining card identification and sRsrcType values from MacDTS

Apple Macintosh Developer Technical Support (MacDTS) can assign card identification and functional sRsrcType values. A HyperCard stack that will help you enter and send the necessary information to MacDTS is available on the *Developer Helper* CD and on AppleLink.

If you don't have access to the HyperCard stack, you must gather the following information:

- the functions the card performs
- the official product name (or code name) for the card
- the driver status (for example, whether the card will have a software driver other than one that has been predefined, such as Apple's video driver)
- the driver location (whether the driver will be on board in ROM or will be installed at initialization time, whether it will be in the application, and so on)
- the company address (postal and electronic mail addresses, if possible) and the name and phone number of the person in the company responsible for the expansion card

With the above information, MacDTS can assign the values for Category, cType, DrSW, and DrHW. All information you provide remains strictly confidential.

Data types

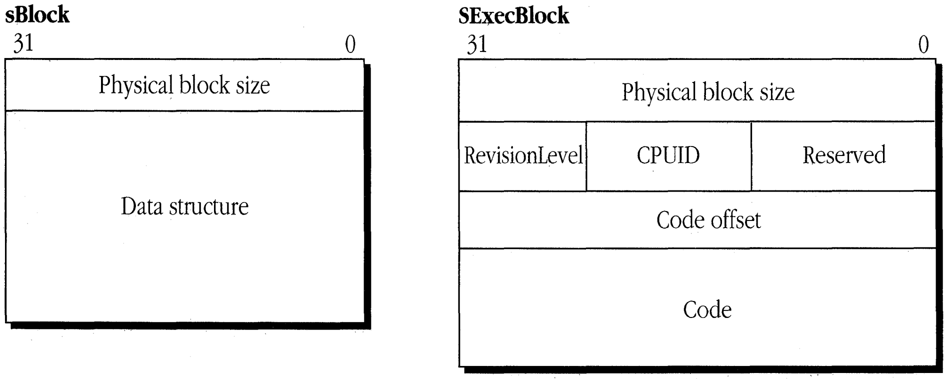
Table 8-1 shows the data types used for communication between the Slot Manager and the card's declaration ROM firmware. Two of the data types are illustrated in Figure 8-3.

■ **Table 8-1** Data types

Data type	Description
`byte`	8 bits, signed or unsigned
`word`	16 bits, signed or unsigned
`long`	32 bits, signed or unsigned
`pointer`	32 bits, signed or unsigned
`cString`	One-dimensional array of bytes, the last of which has the value $00
`offset`	24 signed bits padded to 32 bits, representing a self-relative offset; only bytes in valid byte lanes are counted
`sBlock`	See Figure 8-3
`SExecBlock`	See Figure 8-3

In both examples shown in Figure 8-3, the value of the physical block size field must be the size of that field (4 bytes) plus the actual physical block size. For example, if the data structure in the `sBlock` data type is 100 bytes long, then the value of the physical block size must be 104 bytes. In the example of the `SExecBlock` data type, the `RevisionLevel` field is always 02, the `Reserved` field is always 00, and the `CPUID` field identifies the processor—01 for the 68000, 02 for the 68020, 03 for the 68030, and 04 for the 68040.

■ **Figure 8-3** Formats of `sBlock` and `SExecBlock` data types



- ◆ *Note:* Whenever offset values are used in the declaration ROM firmware, they count only bytes in byte lanes actually being used. Hence these values may be less than the arithmetic difference between the two addresses being offset. For a discussion of byte lanes, see "NuBus Bit and Byte Structure" in Chapter 7.

Firmware structure

This section gives a detailed description of the elements of a generic NuBus card's declaration ROM firmware. If you read the section "An Introduction to the Firmware" in the beginning of this chapter, you should already have a good understanding of what an sResource is and how you use the fields in an `sRsrcType` entry to define the sResource in the firmware. The information on sResources in this section covers the same material but is much more detailed.

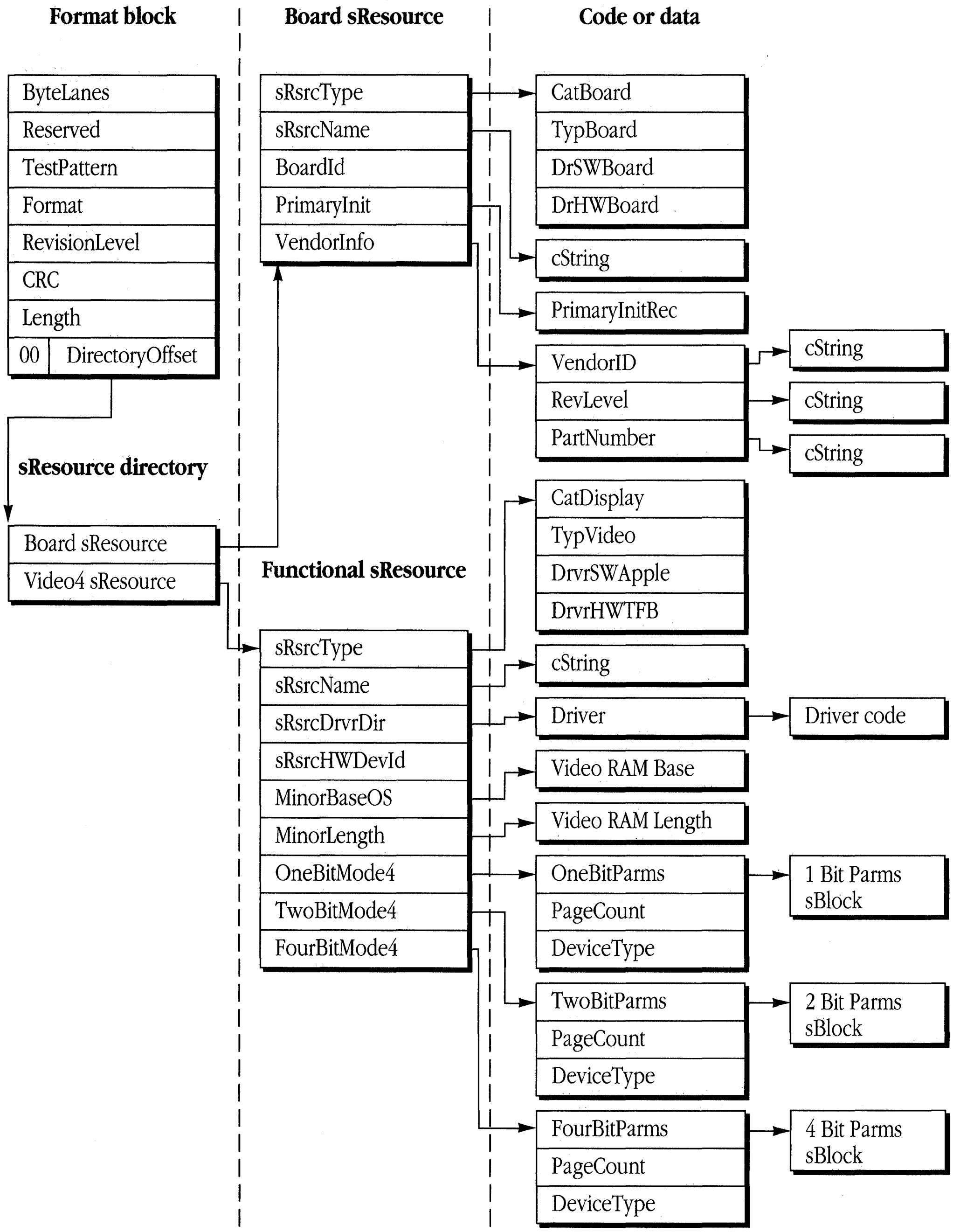
Video cards are more complex than other NuBus cards. They require additional elements in their firmware structure, which are described later in this chapter in the section "Additional Firmware Requirements of Video Cards."

Every NuBus card's declaration ROM firmware must include a format block, an sResource directory, and a board sResource. It should include at least one functional sResource that identifies the card and its function. Figure 8-4 illustrates the relationship of these elements in the declaration ROM of the Macintosh II Video Card. The firmware structure shown in Figure 8-4 is also used in the sample code listing in Appendix B.

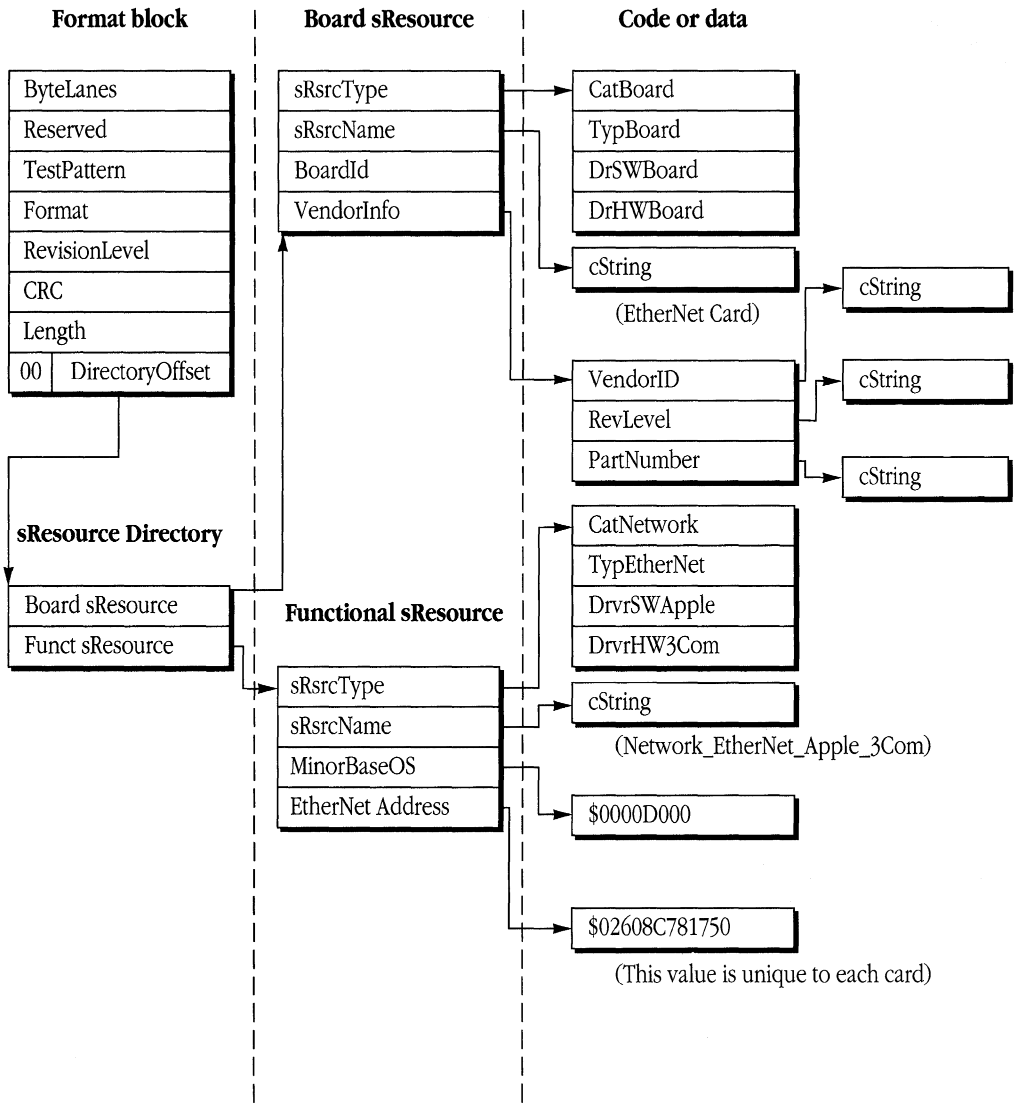
As a comparison, Figure 8-5 shows the simpler firmware structure of the Macintosh II EtherTalk Interface Card.

- ◆ *Note:* To simplify Figure 8-4, only one functional video sResource is shown. However, the Macintosh II Video Card is actually available in two different memory configurations, and the ROM contains a functional video sResource for each configuration. On startup, during the primary initialization routine, the configuration that is not applicable is deleted, and the correct functional sResource is written into the declaration ROM's slot resource table (a data structure that the Slot Manager maintains in memory).

■ **Figure 8-4** Firmware structure of the Macintosh II Video Card



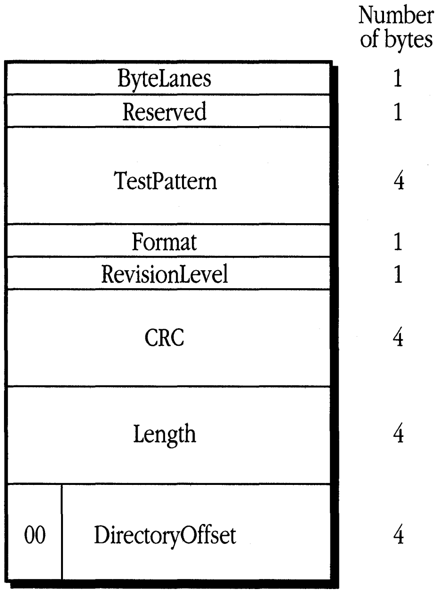
■ **Figure 8-5** Firmware structure of the Macintosh II EtherTalk Interface Card



The format block

The **format block** allows the Slot Manager to find the ROM and validate it. The format block starts at the highest address of the declaration ROM and follows at immediately lower addresses, counting only those addresses accessed by valid byte lanes. Byte lanes are discussed later in this section. The overall format block structure is shown in Figure 8-6.

■ **Figure 8-6** Format block structure



	Number of bytes
ByteLanes	1
Reserved	1
TestPattern	4
Format	1
RevisionLevel	1
CRC	4
Length	4
00 DirectoryOffset	4

The first byte of the format block must reside at one of the 4 bytes at the end of the standard slot space defined under "Slot Allocations" in Chapter 7—that is, the format block must begin with a NuBus address in the range $FsFF FFFF through $FsFF FFFC, where *s* is the slot number. The actual starting address used depends on the value of the `ByteLanes` field, as discussed in the next section.

When the computer is started up, the Slot Manager searches its slots for installed cards, as described in the Device Manager information of *Inside Macintosh.* For each slot it first searches NuBus addresses $FsFF FFFF through $FsFF FFFC (where *s* is the slot number), looking for a valid `ByteLanes` value. If the Slot Manager finds a valid `ByteLanes` value, it verifies this value by examining the `TestPattern` field. Once the Slot Manager verifies the test pattern, it gets the `CRC` (cyclic redundancy check) value and checks the whole ROM to see if it matches the `CRC`. If everything matches, the Slot Manager recognizes the declaration ROM as valid.

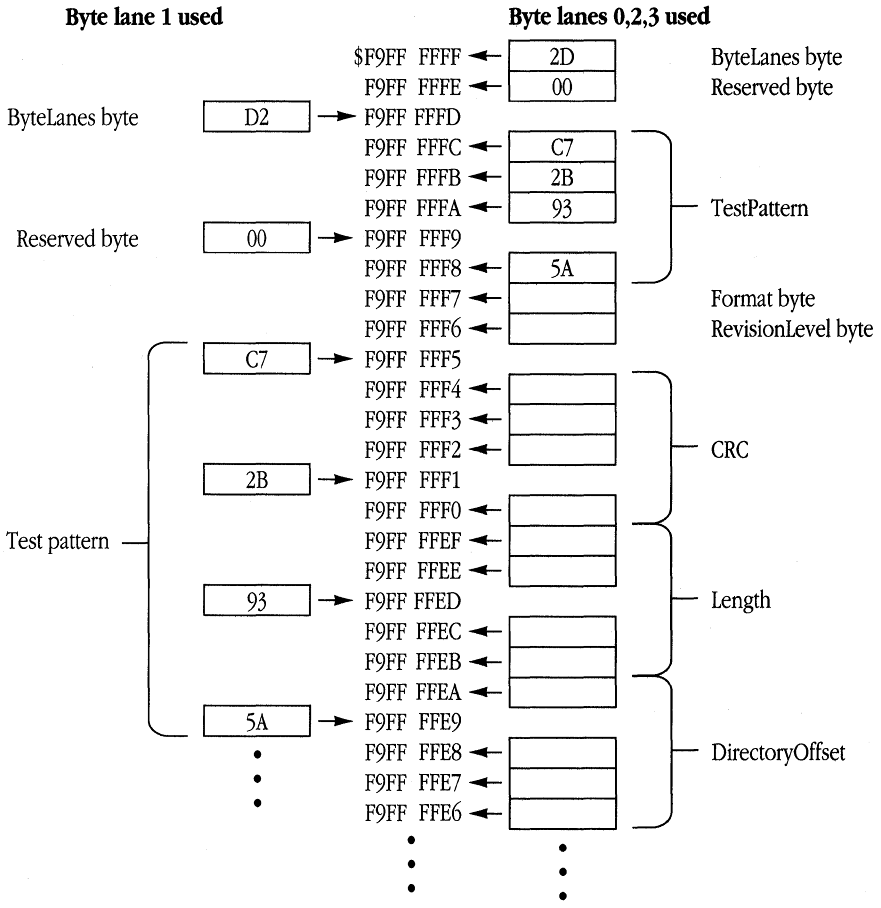
If no valid `ByteLanes` and `TestPattern` values are found, the Slot Manager stores a slot error in the corresponding `sInfo` record, as described in the Slot Manager information in *Inside Macintosh.*

The format block also contains format and identification information and ends with an offset to the sResource directory.

Figure 8-7 shows two examples of the actual structure of a format block, with the addresses that would be used if the card were installed in slot $9. The structure on the left assumes that only byte lane 1 is used; the structure on the right assumes that byte lanes 0, 2, and 3 are used. Typicallly ROM sits only on one byte lane.

The individual fields of the format block are discussed in the sections that follow.

■ **Figure 8-7** Format block examples



ByteLanes

The `ByteLanes` field tells the computer which of the four NuBus byte lanes to use when communicating with an expansion card's declaration ROM. NuBus byte lanes are defined under "NuBus Bit and Byte Structure" in Chapter 7. The value of `ByteLanes` is composed by setting a bit in the low nibble for each byte lane used and then setting the high nibble to the low nibble's complement. The location of the first bit set to 1 in the low nibble also determines the address of `ByteLanes`, and hence the starting address of the format block. Table 8-2 shows all the possible `ByteLanes` values and their corresponding format block starting addresses (where *s* is the slot number). Notice that the `ByteLanes` byte always occupies the highest address available in the byte lanes being used. The Slot Manager doesn't recognize any `ByteLanes` values not shown in Table 8-2.

■ **Table 8-2** Possible ByteLanes values

Bytelanes used	ByteLanes value	Address of ByteLanes
0	$E1	$FsFF FFFC
1	$D2	$FsFF FFFD
0,1	$C3	$FsFF FFFD
2	$B4	$FsFF FFFE
0,2	$A5	$FsFF FFFE
1,2	$96	$FsFF FFFE
0,1,2	$87	$FsFF FFFE
3	$78	$FsFF FFFF
0,3	$69	$FsFF FFFF
1,3	$5A	$FsFF FFFF
0,1,3	$4B	$FsFF FFFF
2,3	$3C	$FsFF FFFF
0,2,3	$2D	$FsFF FFFF
1,2,3	$1E	$FsFF FFFF
0,1,2,3	$0F	$FsFF FFFF

Reserved

The Reserved field must be set to $00.

TestPattern

The TestPattern field identifies the format block. It must be set to $5A93 2BC7.

Format

The 1-byte Format field identifies the declaration ROM format. A Format value of $01 designates the Apple format.

RevisionLevel

The 1-byte RevisionLevel field identifies the current ROM revision. The Slot Manager accepts RevisionLevel values in the range 1–9. RevisionLevel values above 9 cause it to generate a fatal error.

CRC

The 4-byte cyclic redundancy check (`CRC`) value constitutes a checksum to allow the Slot Manager to validate the whole declaration ROM. It is computed by applying a 32-bit rotate-left-and-add function to the number of bytes specified by the `Length` field. Only the bytes specified by the `ByteLanes` field are used to calculate the `CRC` value. For example, if the value of `ByteLanes` is $E1, the calculation would use only the bytes at addresses $FsFF FFFC, $FsFF FFF8, $FsFF FFF4, and so on. In making the `CRC` computation, the value of `CRC` itself is treated as 0. Here is the basic algorithm:

```
        Start pointer at bottom of ROM (top of ROM – Length)
        Initialize sum to 0 (sum will be the calculated CRC value)
@1      Rotate sum left by 1 bit (with ROL.L  #1 instruction)
        If pointer is pointing to the CRC field in the format header, go to @2
        Get the byte pointed to by pointer
        Add the byte to sum
@2      Increment pointer to next data byte
        Go to @1 until done (as specified by Length bytes)
```

Length

The `Length` field contains a long value specifying the number of bytes from the declaration ROM's starting address (as specified by the `ByteLanes` value) to the lowest-address byte of the sResource data structures.

DirectoryOffset

The long `DirectoryOffset` value specifies the self-relative signed offset from the offset itself to the sResource directory. It counts only the bytes in the NuBus byte lanes being used, not the absolute address difference. For example, if the directory address appeared $1000 bytes before the `DirectoryOffset` field in the declaration ROM image and the value of the `ByteLanes` field in the ROM was $E1 (meaning every fourth byte was valid), then `DirectoryOffset` would equal –$1000 even though the directory appears, to the central processor, to be $4000 bytes before the `DirectoryOffset` field. The Slot Manager performs the necessary calculations, based on the byte lanes used, to determine the address of the directory (in this case, it multiplies –$1000 by 4 to get –$4000).

The sResource directory

The **sResource directory** is another major element in a NuBus card's declaration ROM. The sResource directory lists all the sResources in the card firmware and provides an offset (counting only valid byte lane bytes) to each one. Figure 8-4 shows an sResource directory for two sResources: one sResource for a video function and one board sResource.

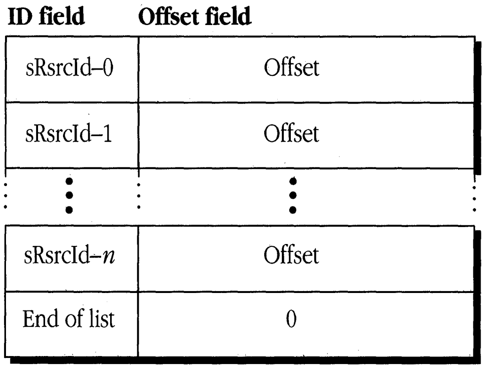
Each sResource defined by a card designer must have a unique identification number in the range 128–254. Identification number 255 is used as an end-of-list marker.

Identification numbers in the sResource directory and in every sResource listed must be in ascending order.

◆ *Note:* Identification numbers in the range 0–127 are reserved for sResources that are defined by Apple for all declaration ROMs. At present there is only one of these: the board sResource, described later in this chapter.

Figure 8-8 shows the sResource directory structure.

■ **Figure 8-8** sResource directory structure



ID field	Offset field
sRsrcId–0	Offset
sRsrcId–1	Offset
⋮	⋮
sRsrcId–*n*	Offset
End of list	0

Each entry in the sResource directory (except the end-of-list entry) points to an sResource. Each entry consists of 32 bits, allocated as follows:

31–24	sResource identification number
23–0	Offset from the entry to the sResource, counting only valid byte lanes

The last entry in the sResource directory must have an offset of 0 and an identification number of 255; that is, it must have the value $FF00 0000.

sResource structure

If you read the section "An Introduction to the Firmware" at the beginning of this chapter, you should already be familiar with sResources. Some of the information is repeated in this section, but the information here is more detailed and may answer questions you have regarding an sResource.

Each sResource contains a number of entries that refer to information about a single capability or function of the expansion card. This information must include the type and name of the resource; it may also include optional items such as the resource's icon, driver, parameters, and so on. The driver is not optional if the sResource is a startup resource or may be a startup video source. Figure 8-4 shows how an sResource defining a video function relates to the other elements in a video card's declaration ROM. The general structure of an sResource is shown in Figure 8-9.

- **Figure 8-9** sResource structure

ID field	**Offset field**
sRsrcType	Offset
sRsrcName	Offset
sRsrcIcon	Offset or data
⋮	⋮
End of list	0

Each entry listed for an sResource must have one of the following three data type forms:

`offset`	Bits 31–24	Identification number
	Bits 23–0	Offset to `long` data, `cString`, `sBlock`, or another list
`word`	Bits 31–24	Identification number
	Bits 23–16	$00
	Bits 15–0	`Word` data
`byte`	Bits 31–24	Identification number
	Bits 23–8	$00 00
	Bits 7–0	`Byte` data

These data types are defined under "Data Types," earlier in this chapter. The last entry listed in every sResource must have the value $FF00 0000. Identification numbers for sResource entries defined by the card designer must be in the range 128–254. They identify items addressed by driver or application code. Identification numbers in the range 0–127 are reserved by Apple; those currently assigned to certain Apple-defined sResource entries are listed in the following sections.

To simplify construction of the sResource structures, Apple has defined two assembly macros, Offset List Entry (`OSLstEntry`) and Data List Entry (`DatLstEntry`). These macros are shown in the sample code in Appendix B.

Apple-defined sResource entries

Table 8-3 lists some of the Apple-defined sResource entries recognized by the Slot Manager, and the sections following describe them. Notice that the `sRsrcType` and `sRsrcName` entries are required; all others are optional. The entries must be listed in ascending numerical order.

■ **Table 8-3** Apple-defined sResource ID numbers

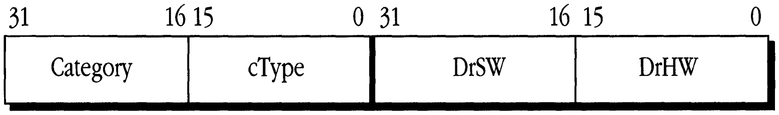
Name	ID number	Description
`sRsrcType`	1	Type of the sResource (required)
`sRsrcName`	2	Name of the sResource (required)
`sRsrcIcon`	3	Icon for the sResource
`sRsrcDrvrDir`	4	Driver directory for the sResource
`sRsrcLoadRec`	5	Load record for the sResource
`sRsrcBootRec`	6	Boot record
`sRsrcFlags`	7	sResource flags
`sRsrcHWDevId`	8	Hardware device ID
`MinorBaseOS`	10	Offset from `dCtlDevBase` to the sResource's hardware base in standard slot space ($Fss0 0000 for 24-bit mode, $Fs00 0000 for 32-bit mode)
`MinorLength`	11	Length of the sResource's address space in standard slot space
`MajorBaseOS`	12	Offset from `dCtlDevBase` to the sResource's hardware base in super slot space
`MajorLength`	13	Length of the sResource in super slot space
`sRsrcCicn`	15	Color icon
`sRsrcIcl8`	16	8-bit icon data
`sRsrcIcl4`	17	4-bit icon data
`sMemory`	108	Resource list for NuBus expansion cards that can act as NuBus masters; defines address ranges from RAM, ROM, and devices

sRsrcType

The type entry in an sResource is used by the Macintosh Operating System or by an application or a driver to identify the function or capability of the sResource. It is a required sResource entry. The actual value of the `sRsrcType` entry is an offset to an 8-byte format defined by Apple. This format is designed to cover all possible devices that might be supported by a card in the computer's expansion slot. However, a bit flag in the format allows the card designer to substitute any other format. The format of the `sRsrcType` entry is shown in Figure 8-10. Remember that the fields in the `sRsrcType` entry have a hierarchical structure.

▲ **Warning** Non-Apple sResource type formats may conflict with each other. If possible, you should use only the Apple format and Apple-assigned values. ▲

■ **Figure 8-10** The `sRsrcType` format



The fields in the Apple `sRsrcType` format are as follows:

`Category` `Category` is the most general classification of card functions. Categories include display, network, terminal emulator, serial, parallel, intelligent bus, and human input devices.

`cType` The `cType` field is a subclass within a category. Within display devices, for example, are video cards and graphics extension processors; within networks, AppleTalk and Ethernet.

`DrSW` The `DrSW` field identifies the driver software interface for the sResource. It also specifies how parameters are stored in the sResource.

`DrHW` The `DrHW` field identifies the hardware associated with the sResource and its driver interface.

The value of each sRsrcType field is unique and is assigned by Apple. To obtain sRsrcType values for the card you are designing, contact Apple Macintosh Developer Technical Support. Refer to the section "Obtaining Card Identification and sRsrcType Values From MacDTS," earlier in this chapter, for details.

sRsrcName

The sRsrcName entry in an sResource provides the name of the sResource. It is a required sResource entry also. The actual value of the entry is an offset to a cString not more than 254 characters long. By convention, the name is derived by stripping the prefixes from the sRsrcType values and separating the fields by underscores. For instance, the sRsrcName field for an sResource whose sRsrcType values are CatDisplay, TypVideo, DrSWApple, and DrHWTFB becomes 'Display_Video_Apple_TFB'. The routine sGetDrvrName prefixes a period to the value of this cString and converts its type to Str255. This name should also be embedded in the driver header of the card's MacOS driver. The _Open routine uses this name to open a driver on disk if one is not present in the ROM, or if the disk version is newer.

sRsrcIcon

The sRsrcIcon entry in an sResource provides the icon for the sResource. The actual value of the entry in the sResource is an offset to a resource of type 'ICON'. This is an optional sResource entry.

sRsrcDrvrDir

An sRsrcDrvrDir or an sRsrcLoadRec entry (described in the next section) is required if the sResource needs a driver to be installed in the Macintosh Operating System before system extensions are called. Otherwise, both are optional. An sRsrcDrvrDir entry is required if the driver for the sResource resides in the card's declaration ROM; an sRsrcLoadRec entry is required if the sResource resides in an external location, such as a hard disk attached to the card.

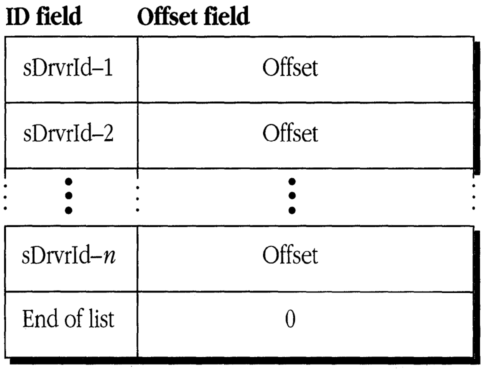
The actual value of the sRsrcDrvrDir entry in the sResource is an offset to an sDriver directory. Each entry in the sDriver directory contains an offset to an sDriver record and an sBlock containing the driver code. The identification number for each entry specifies which operating system supports the driver. Table 8-4 gives the standard sDriver directory identification numbers. Figure 8-11 shows the structure of a typical sDriver directory.

■ **Table 8-4** `sDriver` directory ID numbers

Name	ID number	Description
`sMacOS68000`	1	Driver will run on a Macintosh computer with MC68000 microprocessor
`sMacOS68020`	2	Driver will run on a Macintosh computer with MC68020 microprocessor
`sMacOS68030`	3	Driver will run on a Macintosh computer with MC68030 microprocessor
`sMacOS68040`	4	Driver will run on a Macintosh computer with MC68040 microprocessor

◆ *Note:* Other identification numbers may be used for future Macintosh-family operating systems.

■ **Figure 8-11** Typical `sDriver` directory



ID field	Offset field
sDrvrId–1	Offset
sDrvrId–2	Offset
⋮	⋮
sDrvrId–*n*	Offset
End of list	0

sRsrcLoadRec

Either an `sRsrcLoadRec` entry or an `sRsrcDrvrDir` entry (but not both) is required if the sResource needs a driver to be installed in the Macintosh Operating System before system extensions are called; otherwise, both are optional. The `sRsrcDrvrDir` entry is discussed in the preceding section.

The actual value of the sRsrcLoadRec entry is an offset to an sLoadDriver record. The sLoadDriver record has the format of an SExecBlock and contains the code necessary to load the appropriate driver. The SExecBlock is described under "Data Types," earlier in this chapter.

sRsrcBootRec

The sRsrcBootRec entry in an sResource is an offset to an sBootRecord. The sBootRecord is needed whenever the computer starts from a NuBus card instead of from the internal hard disk or floppy drive. Either the Macintosh Operating System or an entirely different operating system can be installed from a card using sBootRecord.

The sBootRecord has the same format as an SExecBlock. The structure of the SExecBlock is described under "Data Types," earlier in this chapter.

The computer attempts to start from a NuBus card only when certain values are set in its parameter RAM. You can get access to these values by using the Start Manager, as described in the Start Manager information in *Inside Macintosh.*

If an sResource with the specified ID in the specified slot exists, and that sResource has an sBootRecord, it is used for startup. Otherwise, the normal Macintosh startup process occurs.

The sBootRecord code is first called early in the ROM-based startup sequence, before any access to the internal drive. It is passed an seBlock pointed to by register A0. If a non-Macintosh operating system is being installed, the sBootRecord can pass control to it. In this case, control never returns to the normal start sequence in the Macintosh ROM.

When the Macintosh Operating System is started up, the sBootRecord is called twice. The first time, when the value of seBootState is 0, the startup code tries to load and open at least one driver for the card-based device and install it in the disk drive queue. It returns the RefNum of the driver or an error status. That driver becomes the initial one used to install the Macintosh Operating System. During the second call to the sBootRecord, which happens after system patches have been installed but before system extensions have been executed, the sBootRecord must open any remaining drivers for devices on the card.

The sBootRecord can use the _HOpen call to open the driver and install it in the unit table. The _HOpen call will either fetch the driver from the sDriver directory or call the sLoadDriver record if one exists. In any case, the driver's open code must install the driver in the drive queue.

The `sBootRecord` uses the following `SExecBlock` fields:

`seBootState` = 0

→	`seSlot`	The slot number (from PRAM)
→	`seRsrcID`	The sResource ID (from PRAM)
→	`seDevice`	The device number (from PRAM)
→	`sePartition`	The partition number (from PRAM)
→	`seOSType`	Type of operating system to boot (from PRAM)
→	`seReserved`	A reserved field (from PRAM)
→	`seBootState`	0
←	`seRefNum`	Returned `RefNum` of driver to boot with
←	`seStatus`	Returned status (zero = good, negative = no driver loaded)

`seBootState` = 1

→	`seSlot`	The slot number
→	`seRsrcID`	The sResource ID
→	`seDevice`	0
→	`sePartition`	0
→	`seOSType`	Type of operating system (from PRAM)
→	`seReserved`	A reserved field (from PRAM)
→	`seBootState`	1

sRsrcFlags

Two flags are defined in the `sRsrcFlags` word; bit 1, called `fOpenAtStart`, and bit 2, called `f32BitMode`. Bit 1 set (true) tells the Start Manager to install and open the driver at startup time; bit 1 clear (false) tells it to leave the driver closed. Bit 2 set tells the Slot Manager to construct a base address in the form $Fs00 0000; when bit 2 is clear, a base address of $Fss0 0000 results. These base addresses are placed in the `DCE` (device control entry) in the `dCtlDevBase` field. The `DCE` data structure is described in the Device Manager information in *Inside Macintosh*. If there is no `sRsrcFlags` entry, both flags are assumed to be clear (false) by default. All unused flags must be set to 0.

sRsrcHWDevId

The `sRsrcHWDevId` byte entry identifies the sResource as a hardware device. If the sResource is not a hardware device (for example, a data structure), this entry may be omitted. Each hardware device must be given a unique ID.

MinorBaseOS

The `MinorBaseOS` entry contains an offset to a long value that defines the sResource's base address in the slot space allocated to the slot its card is in. The long value is an offset relative to NuBus address $Fs00 0000, where *s* is the slot number. Standard slot space and super slot space are discussed under "NuBus Address Space" in Chapter 7.

MinorLength

The `MinorLength` entry contains an offset to a long value representing the number of bytes of standard slot space occupied by the sResource.

MajorBaseOS

The `MajorBaseOS` entry contains an offset to a long value that defines the sResource's base address in the super slot space allocated to the slot its card is in. The long value is an offset relative to NuBus address $s000 0000, where *s* is the slot number.

MajorLength

The `MajorLength` entry contains an offset to a long value representing the number of bytes of super slot space occupied by the sResource.

sRsrcCicn

The `sRsrcCicn` entry in an sResource provides the color icon for the sResource. To add a color icon, include an `OSLstEntry` with `spID = sRsrcCicn = 15` to your board sResource entries. For more information on color icons, see the section "Icons" later in this chapter. This is an optional sResource entry.

sRsrcIcl8

The optional `sRsrcIcl8` entry provides 8-bit icon data for the sResource. The `'Icl8'` resource is a 32 × 32 × 8-bit color icon in which each pixel is an index into the standard 8-bit system CLUT. A color icon of this form allows full 8-bit color without being penalized by the space overhead requirements of a color table. To add this icon to your board sResource entries, include an `OSLstEntry` with `spID = sRsrcIcl8 = 16`.

sRsrcIcl4

The optional sRsrcIcl4 entry provides 4-bit icon data for the sResource. This 'Icl4' resource is similar to an 'Icl8' resource. It is a compact representation of a 32 × 32 × 4-bit color icon in which each pixel is an index into a standard 4-bit CLUT. You can add this icon to your board sResources by including an OSLstEntry with spID = sRsrcIcl4 = 17.

sMemory

The sMemory resource is a resource list that is provided on "intelligent" Apple NuBus cards capable of being NuBus masters. This resource list provides the address ranges used for RAM, ROM, and device registers for a particular card. This information is used by software to allow intelligent cards in the NuBus to communicate as peers.

An intelligent card is any card with a CPU capable of being a NuBus master and providing memory-like access to the RAM on the NuBus card. The Macintosh Coprocessor Platform (MCP) and boards built upon this architecture will implement the sMemory resource list. It is strongly recommended that the NuBus interfaces on intelligent cards fully support all modes of NuBus access even if multiple local bus cycles are required to complete them. You must not specify the sMemory resource for a NuBus card that does not support all modes of NuBus access.

The sMemory list is currently defined as a second-level resource list. As a second-level resource list, it is not visible to the Slot Manager calls SNextsRsrc and SNextTypeSRsrc, although they are accessible via the advanced-level Slot Manager toolbox calls. In MCP cards, the resource list is an offset from the CPU resource list.

The sMemory resource list contains the required sRsrcType and sRsrcName resources and one or more of the resources listed in Table 8-5.

■ **Table 8-5** sMemory resource list

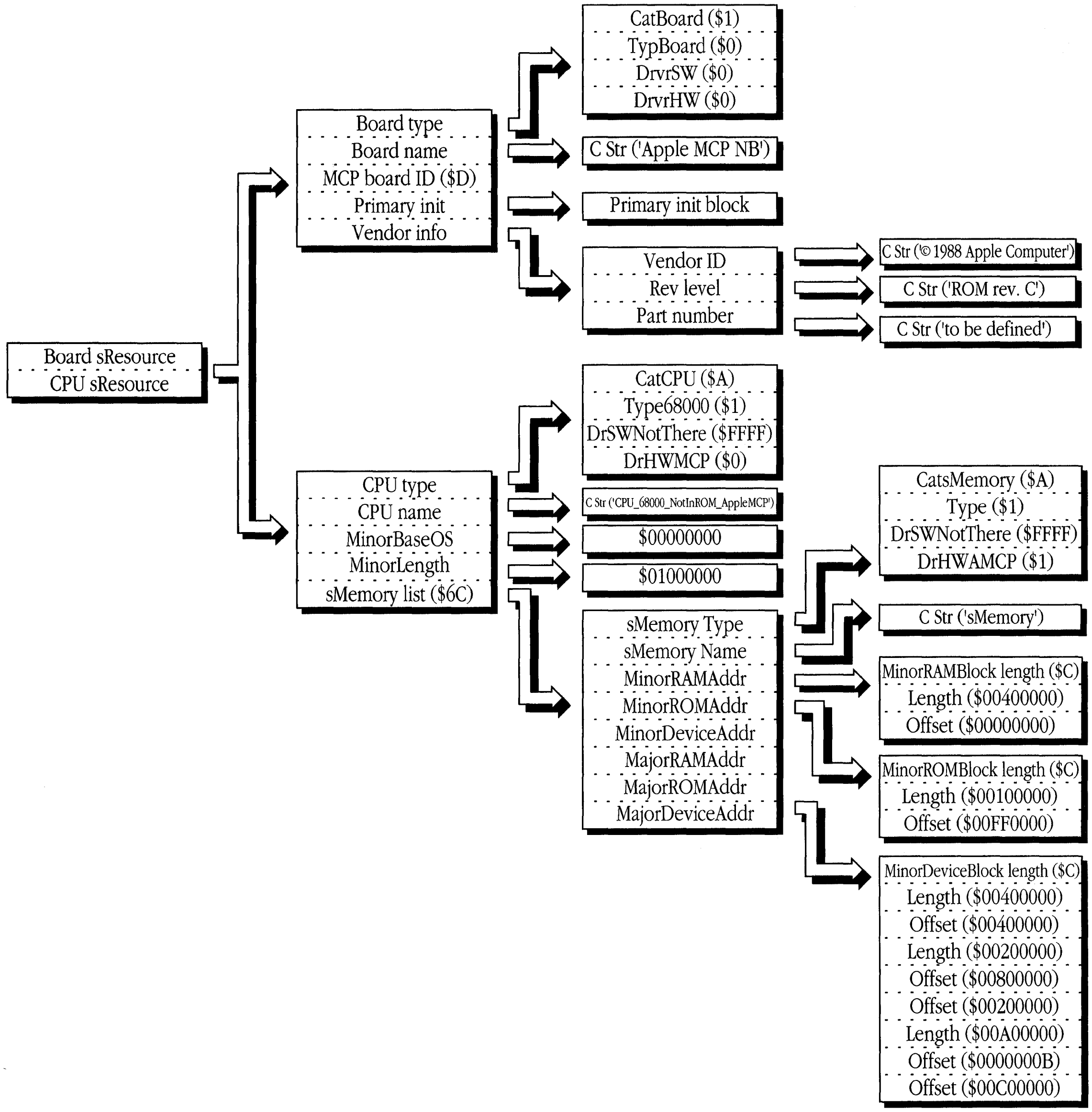
Entry name	ID number	Description
MinorRAMAddr	128	Minor RAM address ranges
MajorRAMAddr	129	Major RAM address ranges
MinorROMAddr	130	Minor ROM address ranges
MajorROMAddr	131	Major ROM address ranges
MinorDeviceAddr	132	Minor device register address ranges
MajorDeviceAddr	133	Major device register address ranges

In each case, the upper 8 bits holds the ID and the lower 24 bits provides an offset to a block. The first longword of the block contains the length of the block followed by pairs of entries. The first longword of each pair has the length of this address range in bytes. The second longword has the offset from the major or minor base address, as appropriate, for this space. See Figure 8-12 for an example of the block format.

The `sMemory` resource list can describe the architected memory structure of the card. It does not have to reflect the actual memory present. For example, if 512 KB of RAM is provided for, but only 128 KB of RAM is present, the resource list can indicate 512 KB of RAM space even though the remaining 384 KB of address space may either mirror the first 128 KB or cause a bus error when accessed. This means that the declaration ROM need not change when memory is expanded. The card must return either a bus error or a data acknowledgment for any memory access within the architected memory range.

Figure 8-12 shows a typical `sMemory` resource list for a generic MCP card.

■ **Figure 8-12** `sMemory` resource list for a generic Macintosh Coprocessor Platform card



The board sResource

If you read the section "An Introduction to the Firmware" at the beginning of this chapter, you are already familiar with a board sResource, a unique sResource that must be present in the firmware of every card that communicates with the computer. This section repeats some of the earlier information, but provides a more in-depth description of the board sResource. Refer to Figure 8-4 to see how the board sResource relates to the other elements in a video card's declaration ROM.

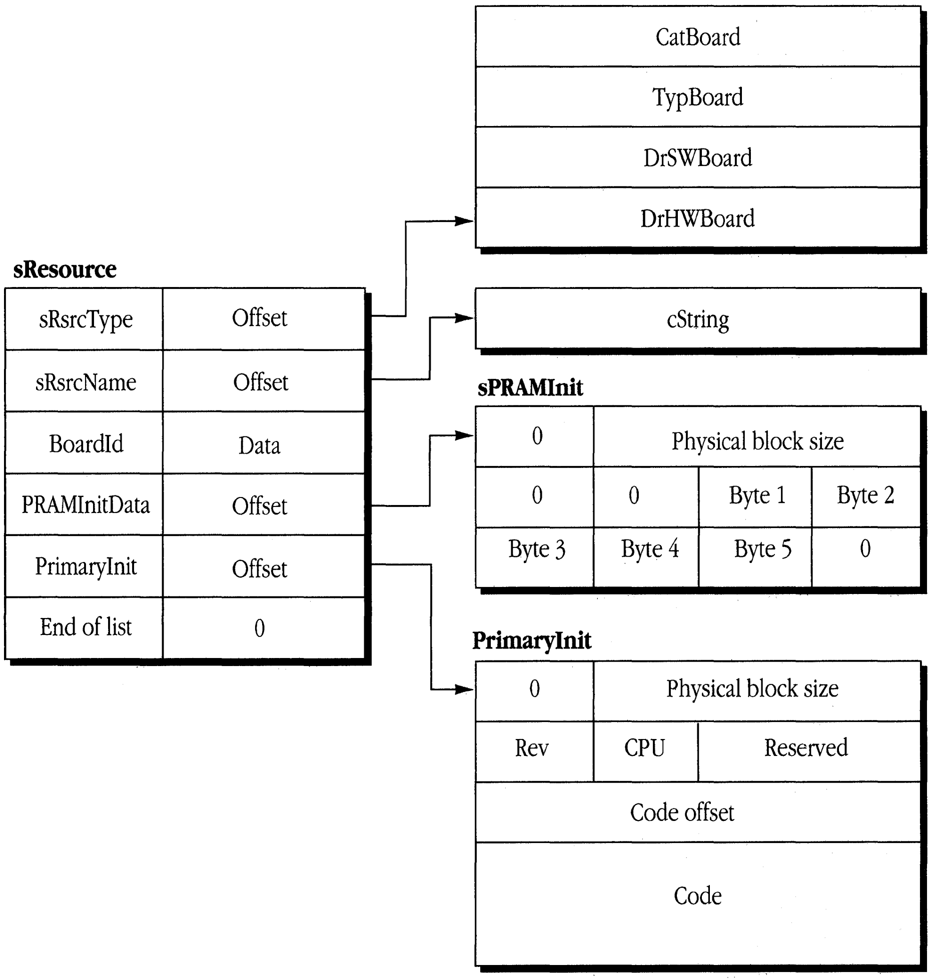
The entries in a board sResource provide the computer with a card's identification number, vendor information, board flags, and initialization code. Table 8-6 lists the standard identification numbers assigned to the Apple-defined entries in the board sResource. These entries are described in detail later in this section.

■ **Table 8-6** Apple-defined board sResource ID numbers

Entry name	ID number	Description
`BoardId`	32	Card design identification number
`PRAMInitData`	33	Data for initializing the PRAM bytes for the slot
`PrimaryInit`	34	Primary initialization code
`STimeOut`	35	`TimeOut` constant
`VendorInfo`	36	Vendor part number, name, and so forth
`SecondaryInit`	38	Secondary initialization code

A board sResource must have entries for `sRsrcType` and `sRsrcName`, which are required for every sResource. Refer to the section "`sRsrcType` Fields for a Video Card Board sResource," earlier in this chapter, for a description of the fields in a board sResource's `sRsrcType` entry. You can also add other Apple-defined sResource entries, such as `sRsrcIcon`. Figure 8-13 shows the structure of a typical board sResource.

■ **Figure 8-13** Typical board sResource



BoardId

`BoardId` is a required entry; without it, the computer will log an error in the appropriate sInfo record. The `BoardId` value is a word (2 bytes) assigned by Apple. To obtain one for the card you are designing, contact Apple Macintosh Developer Technical Support.

PRAMInitData

There are 6 bytes reserved in the parameter RAM (PRAM) of a Macintosh computer for each slot. The `PRAMInitData` entry lets you specify values other than 0 for these bytes. If it is present in the board sResource, the `PRAMInitData` entry provides an offset to an `sBlock` called an `sPRAMInit` record, which contains PRAM initialization values. If it is omitted from the board sResource, the PRAM bytes are initialized to 0. Initialization occurs when the Macintosh Operating System detects a card for the first time or when the Slot Manager finds a `BoardId` entry in a board sResource that is different from the `BoardId` entry in the corresponding `sPRAMInit` record.

The structure of the `sPRAMInit` record is shown in Figure 8-14.

■ **Figure 8-14** `sPRAMInit` record structure

31 24	23		0
0	Physical block size		
0	0	Byte 1	Byte 2
Byte 3	Byte 4	Byte 5	Byte 6

PrimaryInit

The `PrimaryInit` entry contains an offset to a `PrimaryInit` record. The `PrimaryInit` record has the format of an `SExecBlock` containing the code necessary to initialize the card. The structure of the `SExecBlock` is given under "Data Types," earlier in this chapter.

If the `PrimaryInit` record is not present, the computer assumes that the card initializes itself or does not require initialization.

A pointer to an `seBlock` is passed in register `A0` to the `PrimaryInit` code. This parameter block indicates the slot and sResource ID to the `PrimaryInit` code.

You must observe the following restrictions when writing code for the `PrimaryInit` record:

- The code may make no calls to the Macintosh ROM except for Slot Manager routines. In fact, only the Slot Manager is active during `PrimaryInit`.
- The code's length must be less than 16 KB, but ideally should be 2 KB or less.
- The code's execution time should be less than 200 ms.

Initialization code that exceeds these requirements can be placed in the `Open` routine of a driver provided for the card.

In addition to setting up the hardware, `PrimaryInit` must determine the presence of key software components in the computer's ROM. Based on this, the `PrimaryInit` code can determine what sResource lists to retain. If the card's hardware is capable of operating in 24-bit mode, the video sResource lists that describe this mode should be included in case the system is not capable of using the full feature set. In most cases, this involves duplication of the video sResource lists and the deletion of the extended buffer modes. If the `defmBaseOffset` is the same for 24-bit and 32-bit modes, both sResource lists can point to the same `mVidParams` block.

On current Macintosh computers, cards that operate in 24-bit mode should exit `PrimaryInit` with those lists active. On Macintosh computers that include 32-bit QuickDraw in ROM, `PrimaryInit` can exit in either mode, where 32-bit mode would be preferred.

The `PrimaryInit` code is expected to return a status in the `seStatus` field of the `SExecBlock` data structure. This value is saved in the `siInitStatusV` field of the `sInfo` record for the slot. Zero or positive values indicate no error or nonfatal errors. A value of $8001 causes the Slot Manager to defer system initialization of the card until system patches are loaded. It means that 32-bit QuickDraw is not in ROM and that the video card can operate only in 32-bit mode. This forces the Slot Manager to defer using the card as a video device until later, when `SecondaryInit` is run. Negative values indicate that a fatal error occurred while initializing the card; they prevent the Slot Manager from communicating with the card and set an error value in the `siInitStatusA` field of the `sInfo` record.

To summarize, here is the sequence of events that should happen in the `PrimaryInit` code.

- Initialize hardware and disable interrupts. Set up CLUT and gray screen if appropriate.
- Determine the type of display connected, if possible.
- Read parameter RAM (PRAM) to determine if the configuration has changed. For instance, has the RAM configuration changed, or is there a different display?
- Update PRAM to reflect the new configuration.

- Call `_sVersion` to determine which version of the Slot Manager is present in the ROM. If the call returns an error, then the old Slot Manager is in ROM, and `PrimaryInit` should remove any 32-bit addressed sResource lists with `_sDeleteSRTRec`. If no error is returned, the new Slot Manager routines are available in the ROM.
- If the new Slot Manager and 32-bit QuickDraw are in the ROM, remove the 24-bit sResource lists.
- If a valid sResource list is set, return a successful `seResult` value to the Slot Manager. Specifically, if the new Slot Manager is not present, a 24-bit addressed sResource list must be returned. If the new Slot Manager is present, either a 24-bit or 32-bit addressed sResource list can be returned.
- If 32-bit QuickDraw is not in ROM, and the card can only operate in 32-bit addressed mode, return $8001 in `seResult`. This special result code tells the Slot Manager to defer using this card as a video device until later when `SecondaryInit` is run. This device will not display the "Welcome to Macintosh" message or be able to be a MacsBug screen.

STimeOut

The `STimeOut` entry contains the `TimeOut` constant, an option for cards capable of locking out the microprocessor. If the Slot Manager detects a lockout condition, it retries the number of times specified by `TimeOut`. The `sTimeOut` entry is only recognized on the Macintosh II, Macintosh IIx, and Macintosh IIcx computers.

VendorInfo

The optional `VendorInfo` entry in a board sResource contains an offset to a list of `VendorInfo` IDs. These IDs are used only by a vendor and are not assigned by MacDTS. Vendor information should be placed in `cStrings` and use the standard identification numbers shown in Table 8-7.

■ **Table 8-7** `VendorInfo` ID numbers

Name	ID number	Description
`VendorID`	1	The card vendor's design identification
`SerialNum`	2	The individual card's serial number
`RevLevel`	3	The card design's revision level
`PartNum`	4	The part number of the card
`Date`	5	Last revision date of the card

SecondaryInit

On Macintosh computers with version 1 or later of the Slot Manager (this includes the Macintosh IIci, the Macintosh IIfx, and any machine with 32-bit QuickDraw), the SecondaryInit record is executed by the Slot Manager after all system patches have been installed. (The original version of the Slot Manager could not execute SecondaryInit records.) SecondaryInit gives expansion cards a second opportunity to configure their sResources and any other associated RAM structures in case new features were added by the system patch. The rules for SecondaryInit are less stringent than those for PrimaryInit, since the machine is already up and running. Generally, SecondaryInit should focus on performing housecleaning functions on an expansion card's sResources. For example, a video card with direct-mode capabilities cannot be the startup device unless 32-bit QuickDraw is in ROM. This card may determine whether 32-bit QuickDraw is in ROM at PrimaryInit, and, if it is not, it may select an alternative indexed mode that is supported by 32-bit QuickDraw. At SecondaryInit time, and after system patches have been made, the card can again test for 32-bit QuickDraw, and, if it is now present, replace its old video sResource with a new one that includes direct-mode information. In this way, the card can automatically configure itself to the machine environment.

The Slot Manager searches for SecondaryInit records only in those slots that had successful PrimaryInit results. For video cards, a special seResult code ($8001) indicates that no compatible video sResource was selected, but the SecondaryInit should be tried if the new Slot Manager and 32-bit QuickDraw were loaded in the startup process. This allows cards that are only compatible with 32-bit QuickDraw to be used in machines where 32-bit QuickDraw is not in ROM.

Unlike PrimaryInit, SecondaryInit has no size or time limits and executes with system interrupts enabled. Also, SecondaryInit can read and modify pertinent system variables.

Below is a summary of the SecondaryInit activity, specifically for video cards. For more information about the data structures and function calls in this summary, refer to the chapters in *Inside Macintosh* that describe the Slot Manager and graphics devices.

- Call _sVersion to determine which version of the Slot Manager is present in the comuter's ROM. If spResult is returned as 2, the new Slot Manager is in ROM, and all the appropriate configuration has already been done. In this case, return a successful seResult value. If spResult is 1, however, the new Slot Manager was added via a patch, and there is some outstanding cleanup to be done.

- To perform cleanup, execute _GetTrapAddress on the unnamed 32-bit QuickDraw trap ($AB03) and compare it with the address of _Unimplemented. It is possible, though unlikely, that a system may have the new Slot Manager but not 32-bit QuickDraw. If this is the case, leave the 24-bit addressed sResource lists. If the card cannot operate in 24-bit mode, an seResult value indicating failure (–1) should be returned. The card will then be inactive.
- If the card is the startup screen, call _sRsrcInfo to find the driver RefNum. The spRefNum in the parameter block will be 0 if the card is not the startup. The RefNum value may be used during SecondaryInit. If 32-bit QuickDraw is present on your machine, you may need to update the pixel base address in the gDevice structure. The code example in Appendix B demonstrates this in the PrimaryInit and SecondaryInit code.
- Remove the 24-bit version of the sResource list with sDeleteSRTRec.
- With the spRefNum still in the parameter block, verify that spParamData is 0 to enable the sResource list, spID is set to the ID of the sResource list to add, and spsPointer is NIL. Add the 32-bit sResource list using the _sInsertSRTRec function.
- If the card was the startup screen (spRefNum was nonzero), use _GetGDevice to find the gDevice for the card. It will be the only gDevice present at this time. Make sure the device returned by _GetGDevice is your device; it will not be if you returned $8001 from PrimaryInit.
- If the card is the startup screen, set the 32-bit base address in the gpPMap field (gDevice^^.gdPMap^^.pmBaseAddr). If necessary, correct the cached base address in your driver's private storage.
- If the card supports video mode families, use _SetsRsrcState and _sInsertSRTRec to add and enable or disable the appropriate lists.
- Return a successful seResult value.

NuBus block-transfer mode sResource entries

The NuBus '90 draft specification defines two types of block transfers: 1X block transfers and 2X block transfers (or double-rate transfers). Block transfers have only recently been provided in the Macintosh Quadra 700 and the Macintosh Quadra 900 computers. And although the computers in the Macintosh Quadra family allow double-rate block transfers between NuBus cards, they do not support double-rate transfers to or from main memory. Chapter 3 describes block transfers in more detail.

To use the block-transfer capability of NuBus, you must register this capability through a block-transfer sResource.

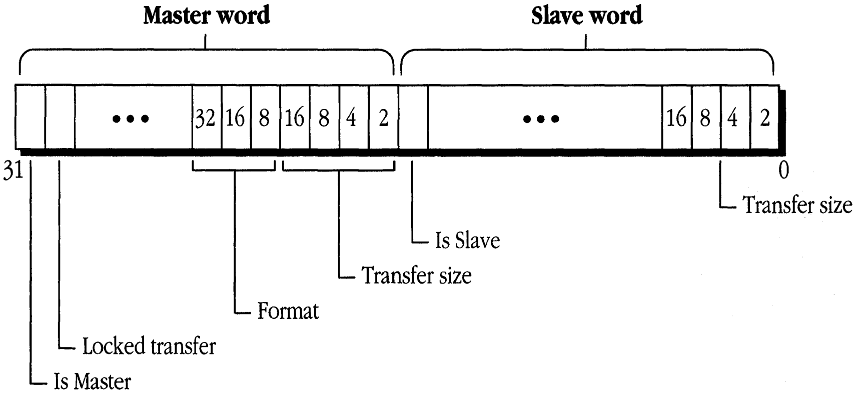
Two longword sResource entries define the block-transfer capabilities of the board or mode. The first describes general block-transfer information (`sBlockTransferInfo`), and the second describes the maximum number of transactions for locked transfers (`sMaxLockedTransferCount`) if the board supports them. If the entries specifying block-transfer information are omitted, the bus master should assume that the target board does not support block transfers and should not test for this capability when the entries are not present.

The second word is not necessary if the board or mode does not support locked transfers.

In the NuBus specification, if a slave board does not support block transfers and if it receives such a request, it should terminate the first transfer with /ACK. Boards that do not support block transfers and do not implement an early /ACK block termination must have the sResource block-transfer information present with the slave transfer size bits set to 0.

The format of the general block-transfer information is a longword whose structure is shown in Figure 8-15. The fields are defined in Table 8-8. The format of the maximum number of transactions for locked transfers is simply one longword.

■ **Figure 8-15** General block-transfer information



■ **Table 8-8** Block-transfer information fields

Field	Value	Meaning
`IsMaster`	1	The board can initiate transfers.
`IsSlave`	1	The board can accept transactions.
`LockedTransfer`	1	The board can initiate locked transfers.
`TransferSize`		Each bit indicates the number of longwords per block transfer; bit set to 1 if the size is supported.
`Format`		Reserved.

The longwords that describe the block-transfer capability are kept in a card's declaration ROM. You can use the `OSLstEntry` (Offset List Entry) macro to describe both block-transfer-capability longwords.

Additional firmware requirements of video cards

The firmware structure of a video card's declaration ROM takes advantage of the power of the Slot Manager. As a result, it is more complex than the declaration ROMs used on most other NuBus cards. It must include data structures that support advanced video functionality and newer drawing systems such as Color QuickDraw. The following sections describe these additions.

◆ *Note:* Color QuickDraw, or 32-bit QuickDraw, is included in the ROMs of some of the Macintosh computers. To determine if Color QuickDraw is present in your system, you must test for it. For more information, refer to the information on compatibility guidelines in *Inside Macintosh.*

Identifying direct devices

The major focus of Color QuickDraw is to support **direct video devices.** A video card is considered a direct device when the pixel value you place in the frame buffer directly implies the color that will appear on the display without going through any intermediate stages of color look-up. Direct video devices have screen depths of 16 bits and 32 bits per pixel.

Prior to Color QuickDraw, Apple supported **fixed video devices** and **indexed video devices** (or CLUT devices) that had screen depths of 1, 2, 4, and 8 bits per pixel. A video card is classified as an indexed device when the values in the frame buffer can be used as an index into a color look-up table (CLUT) to produce an arbitrary color on the display. The color set itself in the indexed video device can be changed. The fixed video device is similar to an indexed video device, except that the hardware colors cannot be changed. The values in the frame buffer can still be used to index any color in the color set, but the color set itself always remains the same.

Setting the `mVidType` field in the video mode parameters to `DirectType` (2) allows Color QuickDraw to determine that a video card is operating in a direct mode (or as a direct device). In addition, the `mVidParams` block of each video sResource should have the following sets of special values for the 16-bit and 32-bit direct modes.

FieldName	**16 bpp**	**32 bpp**
`vpPixelType`	`ChunkyDirect`	`ChunkyDirect`
`vpPixelSize`	`16`	`32`
`vpCmpCount`	`3`	`3`
`vpCmpSize`	`5`	`8`

In the above list, `ChunkyDirect = 16`. The values are the same as those found in a direct pixel-map data structure and are used to construct the `gdPMap` descriptor in `gDevice`.

Identifying 32-bit addressable configurations

The Slot Manager uses a flag in the `sRsrcFlags` word (`spID = 7`) of each sResource to calculate the frame buffer base address for all cards. The sResource flags were defined earlier in this chapter in the section "`sRsrcFlags`." By setting `f32BitMode` (bit 2) in this flag word, all references to the base address of the device are in the form $Fs00 0000, where *s* represents the NuBus slot number of the card. A 24-bit addressed version of the base address in the form of $Fss0 0000 is returned if `f32BitMode` is clear.

The fOpenAtStart flag (bit 1) is normally set at startup time to instruct the Slot Manager to open the slot device's driver at startup time. If this entry is omitted, the field defaults to a value of 2. This default value indicates that the driver should be opened at startup time with a 24-bit-compatible base address (which is the normal condition for traditional video cards). The fOpenAtStart flag must be present for NuBus cards that want to operate in the 16 MB NuBus super slot space.

It is strongly recommended that applications never write directly to the frame buffer. If your application must write directly to the frame buffer, it should operate in 32-bit addressing mode. Previously, cards aliased their frame buffer under 32-bit addressing, as explained in Chapter 1 in the section "NuBus–to–Processor Bus State Machines." If your application must know the addressing state (24-bit or 32-bit), it can determine the device being written to by looking in the gDevice (graphics device) record. For more information, refer to the information about graphics devices in *Inside Macintosh.*

If your Macintosh computer does not have Color QuickDraw in ROM, and if your card is intended as the startup device, make sure that you load Color QuickDraw before the card presents a 32-bit base address to the system.

Icons

You can include manufacturer-specific black-and-white or color icons as an sResource entry. This optional entry was defined earlier in this chapter in the section "sRsrcIcon." The 'ICON' resource defines a black-and-white icon. The 'cicn' resource defines a color icon. If you include either of these resources in the Monitors extension file, the icon is displayed in the Options dialog box of the Monitors control panel. If no icon is found, Monitors displays a generic monitor icon. For more information on Monitors, see the information about the Control Panel in *Inside Macintosh.*

If there is just one function or display associated with your card, an icon can be defined at the board sResource level. However, more often than not, multiple functions or multiple displays are associated with an individual card. For each function or display, a separate icon can be defined. In this case, the icon would be defined at the functional sResource level.

For a black-and-white icon, you can add an OSLstEntry with spID = sRsrcIcon = 3. This entry points to a standard 32 × 32 × 1-bit image of an icon resource. You retrieve this icon by first setting spsPointer to the sResource, spSize to 128, and spResult to a pointer to a 128-byte buffer, and then calling _sReadStruct. There is no mask.

For a color icon, add an `OSLstEntry` with `spID` = `sRsrcCicn` = `15`. Color icons are in `sBlock` form because these structures are variably sized. The offset points to a longword block header that contains the length of the color icon data followed by the image of a standard `'cicn'` resource.

Apple-defined video sResource entries

Table 8-9 lists the Apple-defined video sResource entries recognized by the Slot Manager, and the sections following describe them. These sResources are specific to video cards, and less general in nature than those listed in Table 8-3.

■ **Table 8-9** Apple-defined video sResource ID numbers

Entry name	ID number	Description
`sGammaDir`	64	Gamma directory
`sRsrcVidNames`	65	Video mode name directory

sGammaDir

The `sGammaDir` entry is an optional sResource entry that provides information about gamma resources. It is used only with video cards. You can include this entry in your video sResources by adding an `OSLstEntry` with `spID` = `sGammaDir` = `64`. For more information on the gamma table directory, see the next section, "Gamma Table Data."

sRsrcVidNames

The optional `sRsrcVidNames` entry in a video card's board sResource allows access to the video modes name directory. The video modes names directory identifies the various mode possibilities of video cards that operate in more than one video mode.

Gamma table data

Each functional sResource of a video card can include an optional directory of gamma resources for use with the SetGamma call. This gamma table directory, which is similar in form to the driver directory, permits references to pertinent gamma table data located in the ROM and in the Monitors extension file for the video card. A selector in the Options dialogbox of the Monitors control panel presents the gamma table choices to the user. Monitors loads 'gama' resources from both the extension file and the ROM structure. Ideally, each different monitor supported by the video card uses a specific gamma correction table for the greatest color fidelity. Refer to the section "Gamma Correction in Macintosh Computers," in Chapter 9, for a detailed description of gamma correction.

In each video sResource, an OSLstEntry with spID = sGammaDir = 64 points to a gamma directory. A list of OSLstEntry macros in the directory points to the various gamma table data structures. The gamma tables themselves are sBlock structures—a length field followed by data. The spID values for the gamma tables start at 128, the default gamma for this video mode, and increase by 1 for each optional table present. Since the directory entries are offsets, multiple sResources can point to the same data block. When collecting the data tables, Monitors reads them from the ROM until it encounters an end-of-list entry.

The first field of each gamma block contains 2 ID bytes that are used to localize the name of the gamma table. The next field is a cString format that represents the domestic name of the gamma table. The final field contains an image of the gamma table (equivalent to a 'gama' resource, as defined in *Inside Macintosh*).

Names for gamma tables should reflect the name of the monitor they were intended for, and they must be 35 characters or less in length. The Monitors control panel gamma selector has a checkbox that allows the user to select a gamma table by name or to leave the initial gamma table unchanged. When user selection of gamma is enabled, the control panel creates an additional option, Uncorrected. This option tells the video driver to build a linear gamma table, which in effect disables gamma correction.

Video mode name directory

Video cards that support more than one family of video modes can include an optional directory of names that are used to identify the various mode possibilities. This directory is structured almost identically to the gamma directory.

To access this directory, call _sFindStruct with spID = sRsrcVidNames = 65 in the board sResource (not the functional video sResource). For each possible video sResource spID value there is a similarly numbered OSLstEntry pointing to the name data.

Each name data sBlock contains a 2-byte localization ID followed by a cString containing the name of the video mode. The video mode name should be concise and consist of 35 characters or less.

Video card name

The video card name is visible in the Monitors Options dialog box and must be limited to 35 characters or less.

Resolution

Although QuickDraw does not look at the gDevice resolution fields, new video card designs should set the mHRes and mVRes fields to the approximate characteristics of the monitor.

Sample code

Appendix B provides a sample of the declaration ROM firmware code for the Macintosh II Video Card. This sample uses the Macintosh Programmer's Workshop assembly language. The sample code reflects the configuration of the declaration ROM's firmware shown in Figure 8-4. Also included in the sample code in Appendix B are samples of the primary initialization code and the secondary initialization code for a video card.

Macintosh Coprocessor Platform

When development of various networking and communications products for the Macintosh started at Apple, the need became evident for a real-time operating system that would boost processing power and operating-system capability. As a solution, Apple created an "intelligent" NuBus card, with its own 68000 processor, its own working space in RAM, and its own basic operating-system services. Apple developed this not only as a basis for Apple's own products, but also as a tool for NuBus expansion card developers. The result was the Macintosh Coprocessor Platform (MCP). Its operating system, A/ROSE (Apple Real-time Operating System Environment), was designed to respond to the needs of connectivity products, complement the capabilities of the Macintosh Operating System, and yet be generic enough to become the foundation for a new breed of message-based, distributed software architectures.

Developers can build on this platform in designing products for communications and networking, data acquisition, and signal processing or for other applications that require a substantial amount of processing. Time-consuming and time-critical tasks can be off-loaded from the main logic board to a dedicated processor on the NuBus card. This increases the overall computational speed, of course, and allows for faster response times in the foreground applications. Moreover, A/ROSE provides the real-time, preemptive, multitasking capabilities required for handling multiple communications protocols.

The A/ROSE operating system performs preemptive multitasking, with round-robin task scheduling. It is also a real-time operating system, with 110 µs context switch time and 20 µs of latency (guaranteed interrupt response time). The A/ROSE software can be present on several cards, and it is completely autonomous and independent on each card. Tasks defined by users and by A/ROSE communicate with each other—even across the NuBus to other slots or the Macintosh Operating System—by means of messages.

For more information about the Macintosh Coprocessor Platform or A/ROSE, APDA offers the *Macintosh Coprocessor Platform Developer's Guide.* APDA can also provide development kits if you choose to make use of the Macintosh Coprocessor Platform and the A/ROSE operating system on your NuBus expansion card.

Chapter 9 **NuBus Card Driver Design**

General guidelines for writing drivers are given with the Device Manager information in *Inside Macintosh.* This chapter supplements that information with some specific notes about NuBus card drivers. An example of a video card driver is given in Appendix C to supplement the information provided in this chapter.

Storing the driver code for a NuBus card

You have three choices for storing the driver code for a NuBus card:

- It may be stored as an `sDriver` record in the card firmware. In this case, the driver code is loaded onto the Macintosh system heap immediately before initialization (`'INIT'`) resources are executed, unless specifically inhibited by the `fOpenAtStart` bit in the `sRsrcFlags` field being set to 0. The `sDriver` record is described later in this chapter.
- It can be fetched by an `sLoadDriver` record, in which case the driver code may be stored virtually anywhere. The `sLoadDriver` record is discussed under "`sRsrcLoadRec`" in Chapter 8.
- It may be stored in an initialization resource in the System Folder on a disk that accompanies the NuBus card. In this case, it is installed during system startup as described in the information about the Device Manager in *Inside Macintosh.*

Regardless of where it is stored, a NuBus card driver may be written either for a specific card or for a class of cards. These two approaches are discussed in the following section.

Specific and generic drivers

A NuBus card driver may be written in either of two ways:

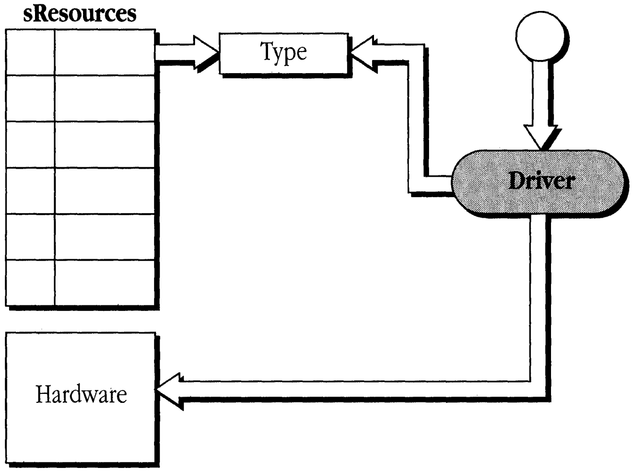
- It may be hard-coded to refer to a specific card.
- It may be written to refer generically to cards of a certain class.

These two approaches are discussed in this section.

Card-specific drivers

A **card-specific driver** contains in its code all the critical information required for it to drive a specific card. For example, if the driver is associated with a video card, it might contain bits-per-pixel information and control register addresses. It could then be used to drive only cards of a specific configuration, as specified by the `sRsrcType` field of the sResource. The way such a driver would work with the card hardware and firmware is diagrammed in Figure 9-1.

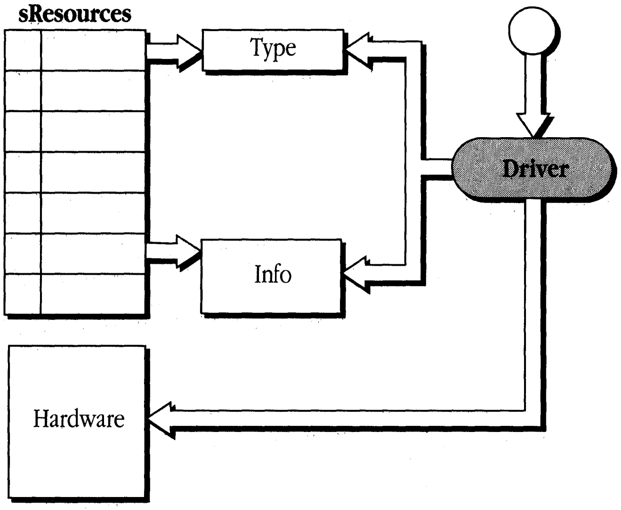
■ **Figure 9-1** Card-specific driver



Card-generic drivers

A **card-generic driver** interrogates the appropriate sResource in the card firmware to determine the hardware configuration with which it must work. sResources are discussed in Chapter 8. For example, a driver associated with a class of video cards might obtain bits-per-pixel information and control register addresses from an sResource in the card's declaration ROM, using Slot Manager calls. The Slot Manager is described in Chapter 8 and in *Inside Macintosh.* The way such a driver would work with the card hardware and firmware is diagrammed in Figure 9-2.

■ **Figure 9-2** Card-generic driver

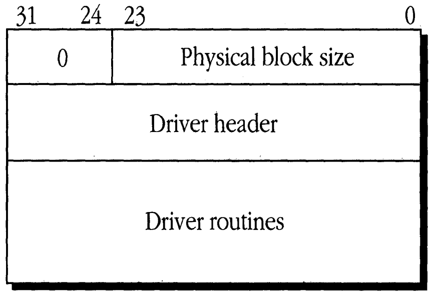


◆ *Note:* You can easily design a video card declaration ROM that supports multiple video devices—for example, devices that work with different types of video monitors. At startup time, all sResources from the sResource directory are loaded into the slot device table. During `PrimaryInit` (and before any screen display), the code can determine the type of monitor connected, delete all other sResources, and run initialization code for the proper display.

The sDriver record

When a driver is stored in the firmware of its associated card, it is placed in an `sDriver` record. An `sDriver` record is a record of type `sBlock`, as defined under "Data Types" in Chapter 8. Its general form is shown in Figure 9-3. The specific structure of the driver header and driver routine sections depends on the operating system with which the driver works. For the Macintosh Operating System, this structure is described in detail with the Device Manager information in *Inside Macintosh.*

- **Figure 9-3** `sDriver` record



Installing a driver at startup

During its startup process, the Macintosh Operating System searches the NuBus slots looking for device drivers to install. As described in Chapter 8, the declaration ROM area of each card contains an sResource directory that points to all the sResources in that card's firmware. Each sResource that refers to a device may contain either actual device driver code or code that allows a driver to be loaded from an external source.

◆ *Note:* The System file may contain drivers for current Apple-designed NuBus cards. Card vendors who supply drivers should use initialization (`'INIT'`) resources to install them during startup. The initialization resources are described with the Start Manager information in *Inside Macintosh.*

For each sResource, the search for drivers during startup takes place as follows:

1. The operating system looks for an `sRsrcFlags` field in the sResource.
2. If no `sRsrcFlags` field exists, or if an `sRsrcFlags` field exists and the field's `fOpenAtStart` bit is set to 1, the operating system searches for a driver, as described in steps 3 and 4. If the value of `fOpenAtStart` is `0`, the operating system does not search for a driver; it goes on to the next sResource.
3. The system searches the sResource for a driver load record (`sRsrcLoadRec`)—a routine designed to copy a driver into the Macintosh system heap. If such a routine exists, the system copies it from the card's ROM to the heap and executes it. The system passes this routine a pointer in `A0` to the `seBlock` structure; on exit, the routine must return a handle in the `seResult` field of the same `seBlock` structure to the driver it has loaded. If the value of the `seStatus` field is 0, the system then installs the new driver.
4. If there is no driver load record, the system searches the sResource for a driver directory entry (`sRsrcDrvrDir`). If there is such an entry and the directory contains a driver of the type `sMacOS68000`, `sMacOS68020`, `sMacOS68030`, or `sMacOS68040`, the system reads the driver from the card's ROM and installs it in the Macintosh system heap. These driver types are downward compatible; an MC68040 CPU can execute `sMacOS68030`, `sMacOS68020`, and `sMacOS68000` drivers as well as the `sMacOS68040` driver.

This method lets you design a card with its driver in ROM on the card. The user can then plug the card in the machine and use the device without running an installation program. Should the driver in ROM later require updating, you can supply an initialization file to be added to the user's System Folder. The initialization file can test the existing driver version and override it with a version contained in its own code, thereby substituting a new driver for the old one.

◆ *Note:* For this method to work correctly, you must follow all the rules for expansion card drivers. In particular, you must include the version number (word aligned) immediately after the driver's name in the driver header structure.

The video driver used at the beginning of system startup (the one that makes the "happy Macintosh" appear) must be taken from a video card's declaration ROM because the System file is not yet accessible. If a system contains multiple video cards, the startup screen is determined by parameter RAM (PRAM) or, if the card specified in PRAM is not present, by selecting a different valid sResource. Note that connecting to a different monitor or changing the amount of frame buffer RAM on a card may cause PRAM to become invalid. If you physically move the card to a different slot, this will cause the PRAM to become invalid as well.

◆ *Note:* As a consequence of the foregoing, any video card that contains the only video device in a system, or supplies the startup device, must have at least a minimal video driver in its declaration ROM.

To install a driver, the ROM first loads it into the system heap and locks it if the `dNeedsLock` bit in the driver flags (`drvrFlags`) word is set. It then installs the driver with a `DrvrInstall` system call and initializes it with an `Open` call. If the driver returns an error from the `Open` call, it is marked closed, the `RefNum` field is cleared in the `ioParameter` block, and the driver is unlocked. Note that this procedure guarantees that driver initialization code will be executed before the system starts executing applications.

Calling a driver

In Macintosh computers, the low-level `PBOpen` routine has been extended to let you open devices in NuBus slots. If the slot serves a single device (not, for example, a chain of disk drives), set the value of `ioFlags` to 0 and use the following parameter block:

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
←	24	ioRefNum	word
→	27	ioPermssn	byte
→	28	ioMix	pointer
→	32	ioFlags	word
→	34	ioSlot	byte
→	35	ioId	byte

In the extension fields, `ioMix` is a long integer reserved for use by the driver `Open` routine. The `ioSlot` parameter contains the slot number of the device being opened, in the range $0–$F (where slot $0 is reserved for built-in video). The `ioId` parameter contains the sResource `spID`.

If the slot serves more than one device, set the value of `ioFlags` to `fMulti` and use the following parameter block:

→	12	ioCompletion	pointer
←	16	ioResult	word
→	18	ioNamePtr	pointer
←	24	ioRefNum	word
→	27	ioPermssn	byte
→	28	ioMix	pointer
→	32	ioFlags	word
→	34	ioSEBlkPtr	pointer

Here the new parameter `ioSEBlkPtr` is a pointer to an external parameter `seBlock` that is customized for the devices installed in the slot. The pointer value is passed to the driver. The `seBlock` structure is described with the Slot Manager information in *Inside Macintosh.*

When a driver serves a device that is plugged into a NuBus slot, it needs to know the slot number, the sResource ID number, and the external device ID number within the slot. The Slot Manager provides values for several new entries on the end of the device control entry (`DCE`) data structure for each sResource. These new entries are

- a byte containing the slot number (`dCtlSlot`)
- a byte containing the sResource ID number for the sResource (`dCtlSlotID`)
- a pointer to the device base address (`dCtlDevBase`) for the driver to use
- a reserved pointer field for future use (`dCtlReserved`)
- a byte containing the external device ID (`dCtlExtDev`)

On a card with multiple instances of the same device, the driver can use `dCtlDevBase` to distinguish among devices. Because the `DCE` address is passed to the driver on every call from the Device Manager, the presence of this pointer in the `DCE` simplifies location of the correct device. This pointer contains the sum of the dynamically determined base address and `MinorBaseOS` or `MajorBaseOS`. (`MinorBaseOS` and `MajorBaseOS` are described under "Apple-Defined sResource Entries" in Chapter 8.) This field is set up before the first call to the driver. The address is always valid in 32-bit mode. The Slot Manager constructs 24-bit or 32-bit compatible addresses based on the `f32BitMode` flag described in Chapter 8. This frees the driver writer from the necessity of locating the hardware for simple slot devices.

Following is the data structure of the DCE. The DCE data structure is described with the Device Manager information in *Inside Macintosh.*

```
; device control entry definition

AuxDCE = PACKED RECORD
   dCtlDriver      DS.L     1           ; ptr to ROM or handle to RAM driver
   dCtlFlags       DS.W     1           ; flags
   dCtlQueue       DS.W     1           ; driver's I/O queue
   dCtlQHead       DS.L     1           ; ptr to first queue element
   dCtlQTail       DS.L     1           ; ptr to last queue element
   dCtlPosition    DS.L     1           ; byte pos used by read/write calls
   dCtlStorage     DS.L     1           ; handle to RAM drivers priv. storage
   dCtlRefNum      DS.W     1           ; RefNum of this driver
   dCtlCurTicks    DS.L     1           ; counter for timing systemTask calls
   dCtlWindow      DS.L     1           ; ptr to driver's window (if any)
   dCtlDelay       DS.W     1           ; number of ticks between systemTask calls
   dCtlMask        DS.W     1           ; desk accessory event mask
   dCtlMenu        DS.W     1           ; menu ID associated with driver
   dCtlSlot        DS.B     1           ; device slot number
   dCtlSlotIdq     DS.B     1           ; device ID within slot
   dCtlDevBase     DS.L     1           ; base address of card for driver
   dCtlOwner       DS.L     1           ; ptr to task control block
   dCtlExtDev      DS.B     1           ; external device ID
                   ALIGN    2
   DevCtlRecEnd    EQU      *-DevCtlRecord          ; size
                   ENDR
```

Slot device interrupts

Slot interrupts from NuBus cards usually enter a hardware register on the computer's main logic board. One interrupt line is dedicated to each NuBus slot connector. The CPU can quickly detect which card requested interrupt service, but not which device on a multifunction card caused the interrupt. To allow proper handling of the interrupt, the Slot Manager provides a slot polling procedure.

The Device Manager maintains an interrupt queue for each slot. Upon receipt of a slot interrupt, the Device Manager goes through the slot's interrupt queue until it gets an indication that the interrupt has been satisfied. If no such indication occurs, an error dialog box, similar to that for system errors, is displayed.

The format for a slot queue element is the following:

```
SQLink          EQU     0       ;link to next element (pointer)
SQType          EQU     4       ;queue type ID for validity (word)
SQPrio          EQU     6       ;priority (low byte of word)
SQAddr          EQU     8       ;interrupt service routine (pointer)
SQParm          EQU     12      ;optional A1 parameter (long)
SQSize          EQU     16      ;length of slot queue element
```

The SQPrio field is an unsigned byte that determines the order in which interrupt routines for a specific card's slot are called. Higher-value routines are called sooner. Priority values 200–255 are reserved for Apple devices.

The SQParm field is a value that is loaded into register A1 before calling an interrupt service routine. This value is set when the driver's interrupt handler is installed as a parameter to SIntInstall. Often, it's useful to pass a handle to the DCE or to the hardware base address (from dCtlDevBase) in this field.

The Device Manager in Macintosh computers provides two new routines to implement the interrupt queue process just described: SIntInstall and SIntRemove.

SIntInstall

```
FUNCTION SIntInstall(sIntQElemPtr: SQElemPtr; theSlot: INTEGER): OsErr;
```

Trap macro _SIntInstall

SIntInstall adds a new element (pointed to by sIntQElemPtr) to the interrupt queue for the slot whose number is given in theSlot. Slots are numbered from $0 to $E. SIntInstall returns an error if it is unsuccessful.

From assembly language, this routine has the following calling sequence:

```
        LEA             MySQE1,A0               ;get slot queue element
        LEA             PollRoutine,A1          ;get routine address
        MOVE.L          A1,SQAddr(A0)           ;set address
        MOVE.W          #Prio,SQPrio(A0)        ;set priority
        MOVE.L          A1Parm,SQParm(A0)       ;save A1 parameter
        MOVE.W          Slot,D0                 ;set slot number
        _SIntInstall                            ;do installation
```

This code causes the routine at label PollRoutine to be called as a result of an interrupt from the specified slot ($0–$E). If two or more slots request an interrupt simultaneously, they are handled in ascending order; that is, within each slot, the interrupt handler with the highest-priority field is handled first.

SIntRemove

```
FUNCTION SIntRemove(sIntQElemPtr: SQElemPtr; theSlot: INTEGER): OsErr;
```

Trap macro `_SIntRemove`

`SIntRemove` removes an element (pointed to by `sIntQElemPtr`) from the interrupt queue for the slot whose number is given in `theSlot`. `SIntRemove` returns an error if it is unsuccessful.

From assembly language, this routine has the following calling sequence:

```
LEA             MySQEl,A0               ;pointer to queue element
_SIntRemove                             ;remove it
```

This routine lets you remove an installed driver containing an interrupt handler from the system without causing a crash.

PollRoutine

Your driver polling routine is called with the following assembly-language code:

```
MOVE.L          SQAddr(A2),A0           ;get poll routine address
MOVE.L          SQParm(A2),A1           ;stuff optional A1 parameter
JSR             (A0)                    ;call polling routine
```

Your polling routine should preserve the contents of all registers except `A1` and `D0`. It should return to the Device Manager with an `RTS` instruction. `D0` should be set to zero to indicate that the polling routine did not service the interrupt or to nonzero to indicate that the interrupt has been serviced. The polling routine should not set the processor priority below 2, and should return with the processor priority equal to 2. The Device Manager resets the VIA2 interrupt flag and executes an `RTE` to the interrupted task when a polling routine indicates that the interrupt is satisfied.

Video drivers

If a NuBus card controls a video display device, there are additional requirements its driver must satisfy. The operating system recognizes that a NuBus card has a video capability by examining the `sRsrcType` fields of its sResources.

To be recognized by the Macintosh system, every video sResource must have an associated driver in the system heap. This driver may be loaded from the card's ROM by the Slot Manager or supplied separately on disk.

Besides using its driver, there are two other ways the system communicates with a video card:

- Its driver must provide a pointer to the card's video RAM, which QuickDraw then accesses directly. Writing pixel information directly into RAM is faster than using driver calls.
- The Slot Manager retrieves information directly from a card's declaration ROM. Such information may include definitions of its potential display modes, as well as data of any kind placed there by the card designer. The declaration ROM data required in video cards is defined in the next section.

Video card firmware normally contains an initialization routine, as described in Chapter 11. The initialization routine should set the video card to a startup mode of 1 bit per pixel, using page 0. It should also clear the video RAM to either the color gray or a 50% gray stipple pattern, and disable vertical retrace interrupts. The Start Manager searches for video sResources, opens the device driver of each card it finds, performs an `InitGDevice` call that sets up the RAM description of the card, and then issues driver calls to set up appropriate screen depth, color table, and other properties.

Each NuBus slot has 8 bytes of dedicated PRAM. The first 2 bytes cannot be modified and always contain the card's `BoardID`. Normally, the other 6 bytes are reserved for the use of the device; but with video devices, the `VendorUse1` field (byte 2) of the slot's PRAM is reserved by the system to hold the `spID` of the slot resource describing the last screen pixel depth that this card was set to. The `InitGDevice` call passes this value to the driver's `SetMode` routine to set the proper hardware pixel depth and uses this value to determine the default color table for this depth. The Monitors Control Panel device sets byte 2.

◆ *Note:* An expansion card's `PrimaryInit` routine should be able to determine whether or not a display is connected at startup time. If no display is connected, the `PrimaryInit` routine removes all video sResources and returns a successful `seResult` code.

Video declaration ROM information

The data structures required in the declaration ROM of any NuBus card are described in Chapter 8. Among them is the sResource, which contains the sResource type, name, and other information about a device. A video sResource should contain a mode list that has a reference for each pixel depth it supports. Such references must begin at ID 128 and continue in ascending order. ID 128 identifies the default mode if a mode is not specified in the sPRAM record. The parameter IDs for mode list entries are shown in Table 9-1.

- **Table 9-1** Video mode list `spID` values

Name	ID number	Description
`mVidParams`	1	Video device record ID.
`mTable`	2	Offset to the device color table for fixed CLUT devices; `mTable` has the same format as the `cTabHandle` structure, described with the Color Manager information in *Inside Macintosh.*
`mPageCnt`	3	Number of video display pages for this mode (expressed as a counting number).
`mDevType`	4	Device type (0 = indexed CLUT device, 1 = indexed fixed device, and 2 = direct device).

The declaration ROM for a video card defines any alternate operating modes for that card. Each mode is completely identified by the following four parameters:

- the number of the slot in which it is installed
- the sResource identification number of the video device it drives
- the identification number of the mode
- the values in its video parameter record

Each distinct mode must have its own video parameter record, with the structure shown in Table 9-2. This structure is the same as the `PixMap` structure described in the Color QuickDraw information in *Inside Macintosh,* except that it describes the physical configuration of a device, not a pixel image.

■ **Table 9-2** Video parameter record

Name	Size	Description
`vpBaseOffset`	`long`	Offset from base of frame buffer to start of page 0
`vpRowBytes`	`word`	Width of each row of video memory (high bit clear)
`vpBounds`	`4 words`	`BoundsRect` for the video display (gives dimensions)
`vpVersion`	`word`	`PixelMap` version number (always 1)
`vpPackType`	`word`	Reserved
`vpPackSize`	`word`	Reserved
`vpHRes`	`fixed`	Horizontal resolution of the display device (pixels per inch)
`vpVRes`	`fixed`	Vertical resolution of the display device (pixels per inch)
`vpPixelType`	`word`	Defines pixel type ($0 = `ChunkyIndexed`; $10 = `ChunkyDirect`)
`vpPixelSize`	`word`	Number of bits in pixel (rounded upward to the next power of 2 for `ChunkyIndexed` and `ChunkyDirect` pixels)
`vpCmpCount`	`word`	Number of components in pixel
`vpCmpSize`	`word`	Number of bits per component
`vpPlaneBytes`	`long`	Reserved

For general information about video card sResource entries, see the section "Apple-Defined sResource Entries" in Chapter 8.

Video driver routines

General instructions for writing device drivers are given in the Device Manager information in *Inside Macintosh.* This section discusses only requirements specific to video drivers.

Normally, a driver associated with a Nubus expansion card may reside either in the card's declaration ROM or on disk. But video drivers differ from other drivers in that they should be able to support screen displays soon after the system is started up, before any code is read from disk. Hence, for video cards, at least a rudimentary driver should reside in the declaration ROM. Such a driver would be loaded during initialization and should display at least 1 bit per pixel. This would let the computer display messages during the startup process.

At a minimum, any video driver must support `Open`, `Close`, control calls, and status calls from the Macintosh Operating System. Your driver's `Open` routine must accomplish the following:

- allocate any private data storage required by the driver
- store a handle to its private data space in the `dCtlStorage` field of the driver's device control entry
- initialize any local variables that the driver uses
- install an interrupt handler for the driver
- enable VBL interrupts on the video card
- determine the configuration of the machine it is running on. This should be done by reading the PRAM, assuming that the configuration is stored in PRAM during `PrimaryInit`. Refer to the Slot Manager description in *Inside Macintosh* for more information.

The operating system does not expect your driver's `Open` routine to set or change the video mode. The Start Manager explicitly sets the appropriate video mode during startup time as determined by PRAM or by an `'scrn'` resource (described in the information on Color QuickDraw in *Inside Macintosh*).

◆ *Note:* All data and flags used by the driver should be stored in the `dCtlStorage` handle rather than in the driver code segment.

Your video driver's `Close` routine must accomplish the following:

- disable VBL interrupts on the video card
- remove the interrupt handler used by the driver, replacing any changed interrupt vectors
- release any private data storage held by the driver
- turn off the video to avoid the persistence of the desktop image, especially during reboots. If your video driver does not explicitly turn off the video, by disabling the sync signal, for instance, you should return your display to its off state (white with no backlighting on LCD-type displays or black with no pedestal on CRT-type displays).

◆ *Note:* Like other Macintosh drivers, video drivers need to expect that they will be closed and reopened (possibly at times other than boot time). For this reason, the `Close` routine should physically and logically turn off the video for the display that the video driver controls. Similarly, the `Open` routine must not assume that `PrimaryInit` has just been run.

Video driver data structures

The Macintosh Operating System communicates with each video driver by means of control and status calls that use the following data structures:

```
TYPE
VDParamBlockPtr = ^VDParamBlock;
VDParamBlock = RECORD
              qLink: QElemPtr;             {standard I/O param block}
              qType: INTEGER;
              ioTrap: INTEGER;
              ioCmdAddr: Ptr;
              ioCompletion: ProcPtr;
              ioResult: OSErr;
              ioNamePtr: StringPtr;
              ioVRefNum: INTEGER;
              ioRefNum: INTEGER;
              csCode: INTEGER;             {video driver specifics}
              csParam: Ptr;
              END;

VDEntRecPtr = ^VDEntryRecord;
VDEntryRecord = RECORD
              csTable: Ptr;                {pointer to color table}
              csStart: INTEGER;            {start entry number}
              csCount: INTEGER;            {count number}
              END;

VDGamRecPtr = ^VDGammaRecord
VDGammaRecord = RECORD
              csGTable: Ptr;               {pointer to gamma table}
              END;

VDPgInfoPtr = ^VDPgInfo;
VDPgInfo = RECORD
              csMode: INTEGER;             {mode within device}
              csData: LONGINT;             {data supplied by driver}
              csPage: INTEGER;             {page to switch in}
              csBaseAddr: Ptr;             {base address of page}
               END;
```

```
VDFlagPtr = ^VDFlagRec;
VDFlagRec = RECORD
            flag: SignedByte;           {used in various ways}
            END;

VDDefModePtr = ^VDDefModeRec;
VDDefModeRec = RECORD
            spID: SignedByte;           {spID}
            END;
```

◆ *Note:* Video drivers follow the newest convention for returning status call information. This convention may not be compatible with the glue code used in previous development systems. For example, using the old convention, results from a `_Status` call were returned directly in the I/O parameter block. Using the new convention, results from the `_Status` call are returned directly to `csParamBlock`. However, in certain cases where `_Status` is called, the glue code neglects to fill in the `csParam` field of the parameter block. If you are interfacing to a device driver that requires `csParam` for its status call, use the lower-level call `PBStatus`. It will return a valid `csParam` field.

Slot information applicable to the card associated with your video driver is contained in the device control entry, as described in the Device Manager information in *Inside Macintosh.*

Control routines

The Macintosh Operating System uses control calls to your video driver to set the video card to different configurations. Configuration changes might include choosing a different number of bits per pixel or changing the color table.

Video driver routines that respond to these control calls are described in this section. The calls that all drivers must support are so identified; others are optional and may return a `NoErr` code.

Throughout this section, you see references to video devices that operate in indexed pixel mode (commonly called *indexed video devices*), devices that operate in direct pixel mode (commonly called *direct video devices*), and devices that operate in fixed indexed pixel mode (commonly called *fixed video devices*). The section "Additional Firmware Requirements of Video Cards" in Chapter 8 describes indexed, direct, and fixed video devices and explains the differences between them.

◆ *Note:* If a specific driver has other hardware capabilities and you want to provide a driver interface to them, you should give these control routines `csCode` selectors greater than or equal to 128.

csCode = 0	**csParam**	**= VdPgInfoPtr**	**[Reset]**
←	`csMode`	mode selected	`[word]`
←	`csPage`	page after reset	`[word]`
←	`csBaseAddr`	base address of video RAM	`[long]`

This required control routine must reset the video card to its startup state. The startup state of a video card should be its default pixel depth (preferably 1 bit per pixel), with the default colors (if colors are supported) set. If the card supports multiple video pages in the default mode, page 0 should be switched in.

Your driver should also reinitialize its private storage areas, including areas for returned parameters.

csCode = 1 **[KillIO]**

This required control routine stops any I/O requests currently being processed and removes any pending I/O requests. For most video cards, no change on the card is required. If the card does not support asynchronous calls, this routine may return a `NoErr` code.

csCode = 2	**csParam**	**= VDPgInfoPtr**	**[SetMode]**
→	`csMode`	mode within device	`[word]`
→	`csPage`	desired display page	`[word]`
←	`csBaseAddr`	base address of video RAM	`[long]`

This required control routine sets the pixel depth of the screen. To improve the screen appearance during mode changes, devices with settable color tables should set all entries of the CLUT to 50% gray. If the video card supports 16-bit or 32-bit pixel depths, this routine should set an internal flag to indicate direct mode operations.

◆ *Note:* QuickDraw requires that all screen depths have the same frame buffer base address.

The Monitor `cdev` stores the current video mode in the card's slot PRAM.

csCode = 3	**csParam**	**= VDEntRecPtr**	**[SetEntries]**
→	`csTable`	pointer to color specification array	`[long]`
→	`csStart`	first entry in table	`[word]`
→	`csCount`	number of entries to set	`[word]`

If the video card is an indexed device, this optional control routine should change the contents of the card's CLUT. If the card does not have a look-up table, it will never receive this call. If the value of `csStart` is 0 or positive, the routine must install `csCount` entries starting at that position. If it is –1, the routine must access the contents of the `Value` field in `csTable` to determine which entries are to be changed. Both `csStart` and `csCount` are zero based; their values are 1 less than the desired amount. For a description of the structure of a CLUT, refer to the information on Color QuickDraw in *Inside Macintosh.* The `SetEntries` control routine is also described in the Color Manager chapter of *Inside Macintosh.*

◆ *Note:* The `csStart` value refers to logical position, not physical position. In 4-bits-per-pixel mode, for example, `csStart` values will still run 0,1,2,..., even though physical card registers may not have this numbering sequence.

If the video card is a direct device, the system should never issue this call, but if it does, `SetEntries` should return an error indication. If a direct device contains CLUT hardware, the `GrayScreen` and `SetGamma` routines are responsible for setting the hardware up properly.

In the 16-bit and 32-bit video modes associated with direct devices, the display color is implied directly by the pixel value. Logically, the three DAC channels in the hardware are completely independent and assumed to be ascending linear ramps in all channels. Since the effect of the `SetEntries` routine (in the Color Manager) is to modify the QuickDraw drawing environment, the `SetEntries` call has no meaning to a direct device.

◆ *Note:* The `SetEntries` control routine is a low-level function and should only be used in special cases. You may find it easier to implement an equivalent higher-level call. Refer to Device Manager and Color Manager information in *Inside Macintosh* for information about alternate function calls.

csCode = 4	**csParam**	**= VDGamRecPtr**	**[SetGamma]**
→	`csGTable`	pointer to gamma table	`[long]`

This optional control routine sets a **gamma table** in the driver that corrects RGB (red, green, blue) color values. The gamma table compensates for nonlinearities in a display's color response by providing either a function or a look-up value that associates each displayed color with an absolute RGB value. The gamma table is described with the graphics devices information in *Inside Macintosh.* Gamma correction is defined and explained later in this chapter in the section "Gamma Correction in Macintosh Computers."

To reduce visible flashes due to color table changes, the `SetGamma` call works in conjunction with a `SetEntries` call on indexed devices. The `SetGamma` call first loads new gamma correction data into the driver's private storage, and then the next `SetEntries` call applies the gamma correction as it changes the CLUT. If the hardware performs gamma correction externally to the CLUT hardware, then the `SetGamma` call should take effect immediately. `SetGamma` calls are always followed by `SetEntries` calls.

For direct devices, `SetGamma` first sets up the gamma correction data table. Next, it synthesizes a black-to-white linear ramp in RGB. Finally, it applies the new gamma correction to the ramp and sets the data directly in the hardware. Proper gamma correction is particularly important to image-processing applications running on direct devices.

Displays that do not use gamma table correction tend to look oversaturated and dark. Although determining the correct values for a gamma table can be difficult without special tools, the table's contribution to image quality can be striking.

If `NIL` is passed for the `csGTable` value, the driver should build a linear ramp in the gamma table to allow for an uncorrected display.

csCode = 5	**csParam**	**= VDPgInfoPtr**	**[GrayScreen]**
→	`csPage`	page number	`[word]`

This optional control routine should fill the specified video page with a dithered gray pattern in the current video mode. The page number is zero based.

The purpose of this routine is to eliminate visual artifacts on the screen during mode changes. When an application changes the screen depth, the contents of the frame buffer immediately acquire a new color meaning. To avoid annoying color flashes, the `SetMode` control call (first in the depth change sequence) sets the entire contents of the CLUT to 50% gray, so that all possible indexes in either the old or new depth appears the same. This routine is called to fill the frame buffer with the new 50% dither pattern. In the last step of the mode change sequence, the color table is filled, making the 50% dither pattern visible.

For direct video devices, `GrayScreen` also builds a three-channel linear gray color table, and after the table has been gamma corrected, it loads it into the color table hardware. The base address is determined by the system software configuration. For example, if 32-bit QuickDraw is present, the base address may be a 32-bit address. If your card is used in an earlier system that does not include 32-bit QuickDraw, the base address is a 24-bit address. To simplify the code, you should always write `GrayScreen` to the screen in 32-bit addressing mode.

csCode = 6	**csParam**	**= VDFlagPtr**	**[SetGray]**
→	`csMode`	mode value	`[byte]`

This optional control routine is used with indexed devices to determine whether the control routine with `csCode` = `3` (`SetEntries`) fills a card's CLUT with actual colors or with the luminance-equivalent gray tones. For actual colors (the default case), the control routine is passed a `csMode` value of 0; for gray tones it is passed a `csMode` value of 1.

Luminance equivalence should be determined by converting each RGB value into the hue-saturation-brightness system and then selecting a gray value of equal brightness. Mapping colors to luminance-equivalent gray tones lets a color monitor emulate a monochrome monitor exactly.

If the `SetGray` call is issued to a direct device, it sets the internal mapping state flag and returns a `CtlGood` result but does not cause the color table to be luminance mapped. Short of using the control routine `DirectSetEntries`, there is no way to preview luminance-mapped color images on the color display of a direct device.

csCode = 7	**csParam**	**= VDFlagPtr**	**[SetInterrupt]**
→	`csMode`	enable/disable flag	`[byte]`

This optional routine controls the generation of the VBL interrupts. To enable interrupts, pass a `csMode` value of 0; to disable interrupts, pass a `csMode` value of 1.

csCode = 8	**csParam**	**= VDEntRecPtr**	**[DirectSetEntries]**
→	`csTable`	pointer to color table	`[long]`
→	`csStart`	first entry in table	`[word]`
→	`csCount`	number of entries to set	`[word]`

Normally, color table animation is not used on a direct device, but there are some special circumstances under which an application may want to change the color table hardware. This routine provides the direct device with indexed mode functionality identical to the regular `SetEntries` call. The `DirectSetEntries` routine has exactly the same functions and parameters as the regular `SetEntries` routine, but it works only on a direct device. If this call is issued to an indexed device, it should return a `CtlBad` error indication.

△ **Important** The application that calls the `DirectSetEntries` routine is responsible for restoring the triple-linear-ramp direct-color environment when it completes the color table animation. △

The `DirectSetEntries` routine is implemented separately from the regular `SetEntries` routine to prevent applications that get direct access to the driver from indiscriminately changing the hardware and rendering the system unusable.

csCode = 9	**csParam**	**= VDDefModePtr**	**[SetDefaultMode]**
→	`csID`	`spID` of video sResource	`[byte]`

A video card may support different configurations for a single display device. For example, a card may support large or small screen sizes on a single monitor. When a card supports different configurations for a single display device, it is said to have a video mode family. Having a video mode family is different from supporting two different monitors, since all members of the family can be displayed on a single display device. The Slot Manager (version 1 and later) supports both video mode families and multiple display devices.

The `SetDefaultMode` routine is used by both indexed and direct devices to specify the selected member of a video mode family for the next restart. It does this by storing the `spID` value of the new choice's sResource in the card's slot PRAM. The Monitors control panel makes the `SetDefaultMode` call when selecting a new video mode. Monitors searches for all video sResources associated with the card, both active and inactive, to create a list of selections. (Note that all video sResources associated with other displays are deleted at this time, but not inactivated.) After selecting a new video mode, Monitors calls `SetDefaultMode` with the new selection's `spID` value as the parameter. The routine stores this value somewhere in its slot PRAM, making this the new default configuration. Monitors also collects the appropriate information from the sResource to construct a valid `'scrn'` resource and takes care of all additional validation that is necessary to make the new mode take effect at the next restart.

At `PrimaryInit` time, the code should determine whether the `spID` value saved in slot PRAM is compatible with the current environment. If it is, it should make the mode the active `gDevice`.

◆ *Note:* On machines that do not have 32-bit QuickDraw in ROM, cards that can be addressed in both 24-bit and 32-bit addressing modes may have to store additional information in PRAM in order to remember the default correctly.

This routine records the default mode information in the slot's private PRAM. Remember that the `VendorUse1` byte is reserved for system use, but the other 5 bytes are available for the private use of the card software.

Status routines

The Macintosh Operating System sends status calls to your video driver to determine the current configuration of the video card.

Video driver routines that respond to these calls are described in this section. The driver need process only pertinent status calls; others it can return with a status error.

◆ *Note:* If your driver supports other devices and you want to provide a driver interface to them, you should give these status routines `csCode` selectors greater than 128.

The `csCode` values 0 and 1 are not implemented in video drivers and should return a `StatBad` result code.

csCode = 2	**csParam**	**= VDPgInfoPtr**	**[GetMode]**
←	`csMode`	mode within device	`[word]`
←	`csPage`	display page	`[word]`
←	`csBaseAddr`	base address of video RAM	`[long]`

This required status routine must return the current video mode, page, and base address.

csCode = 3	**csParam**	**= VDEntRecPtr**	**[GetEntries]**
↔	`csTable`	color table data	`[long]`
→	`csStart`	first entry in table	`[word]`
→	`csCount`	number of entries to set	`[word]`

This required status routine must return the specified number of consecutive CLUT entries, starting with the specified first entry. If gamma table correction is used, the values returned may not be the same as the values originally passed by `SetEntries`. If the value of `csStart` is 0 or positive, the routine must return `csCount` entries starting at that position. If it is –1, the routine must access the contents of the `Value` fields in `csTable` to determine which entries are to be returned. Both `csStart` and `csCount` are zero based; their values are 1 less than the desired amount.

Although direct video modes do not have logical color tables, the `GetEntries` status routine should continue to return the current contents of the CLUT, just as it would in an indexed video mode.

◆ *Note:* The `GetEntries` control routine is a low-level function and should only be used in special cases. You may find it easier to implement an equivalent higher-level call. Refer to Device Manager and Color Manager information in *Inside Macintosh* for information about alternate function calls.

csCode = 4	**csParam**	**= VDPgInfoPtr**	**[GetPages]**
←	`csPage`	number of pages	`[word]`
→	`csMode`	mode within device	`[word]`

This required status routine must return the total number of video pages available in the current video card mode (not the current page number). This is a counting number (not zero based).

csCode = 5	**csParam**	**= VDPgInfoPtr**	**[GetBaseAddr]**
→	`csPage`	desired page	`[word]`
←	`csBaseAddr`	base address of that page	`[long]`

This required status routine must return the base address of a specified page in the current mode. This allows video pages to be written to even when not displayed.

csCode = 6	**csParam**	**= VDFlagPtr**	**[GetGray]**
←	`csMode`	mode within device	`[byte]`

This required status routine must return a value indicating whether the `SetEntries` routine has been conditioned to fill a card's CLUT with actual colors or with the luminance-equivalent gray tones. For actual colors (the default case), the value returned by `csMode` is 0; for gray tones it is 1. The value returned can be set by a control call with `csCode = 6`.

csCode = 7	**csParam**	**= VDFlagPtr**	**[GetInterrupt]**
←	`csMode`	enable/disable flag	`[byte]`

This optional status routine returns a value of 0 if VBL interrupts are enabled and a value of 1 if VBL interrupts are disabled.

csCode = 8	**csParam**	**= VDGamRecPtr**	**[GetGamma]**
←	`csGTable`	pointer to gamma table	`[long]`

This status routine returns a pointer to the current gamma table. The calling application cannot preallocate memory because of the unknown size requirement of the gamma data structure.

csCode = 9	**csParam**	**= VDDefModePtr**	**[GetDefaultMode]**
←	`csID`	`spID` of video sResource	`[byte]`

This status routine returns the current default value of a video sResource's `spID` entry. If you have selected a new mode family, but have not yet rebooted the system, the default returned will be different from that of the current video sResource. The parameter block is the same as for the `SetDefaultMode` control call.

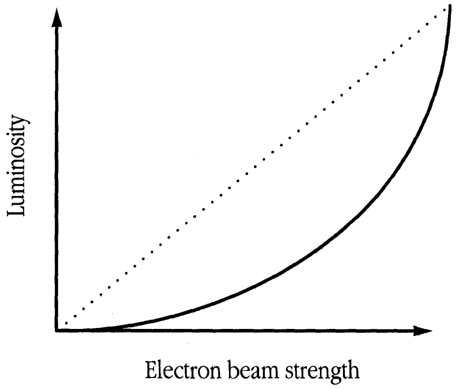
Gamma correction in Macintosh computers

Color QuickDraw considers all colors specified by application programs as absolute specifications; that is, from the application's point of view, a single color specification appears as a uniform color across multiple display devices that have different color responses. Macintosh computers operate with many different display screens. Since the application cannot recognize the different screens and does not have the opportunity to perform screen-by-screen corrections, the video driver for each display device configured in the system must linearize the differences in color (or gray-scale) response. This is called **gamma correction.**

How gamma correction works

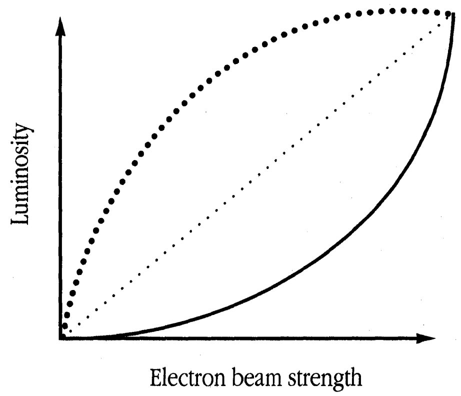
As the beam from a video display's electron gun sweeps the scan lines, it strikes phosphors on the face of the monitor tube and causes them to luminesce. If you increase the intensity of the beam, the phosphor dots luminesce more brightly, and if you reduce the intensity of the beam, the phosphor dots glow less brightly. Unfortunately, the luminescence output of the phosphor dots is not directly proportional to the impinging beam strength but more closely resembles the diagram in Figure 9-4.

- **Figure 9-4** Color response without gamma correction



In this drawing, the dotted line shows the ideal linear response, and the solid line approximates the observed response of a typical phosphor. This curved response characteristic is due to physical phenomena and without gamma correction would cause the colors on the screen to appear darker than expected. Based on this behavior, you can apply an inverse gamma correction function that compensates for the nonlinear response. Figure 9-5 illustrates color response with gamma correction.

■ **Figure 9-5** Color response with gamma correction



In Figure 9-5 the solid line is again the observed response of the phosphor, the heavy dotted line is the inverse gamma function, and the light dotted line is the linear color response that results from the gamma correction.

Gamma correction can be performed by dedicated hardware. As an alternative, it can be performed in the CLUT hardware by substitution in the `SetEntries` call. A number of high-order bits are extracted from the red, green, and blue channels of the required colors and used as an index into a table of corrected values. These values are then placed into the hardware to yield the corrected output. The Macintosh II Video Card uses the high 8 bits of each channel to reference the gamma table.

The gamma table data structure

The following is the structure that supports gamma correction.

```
record GammaTbl of
     gVersion       :integer;                  {gtab version, currently 0}
     gType          :integer;                  {drHWId value}
     gFormulaSize   :integer;                  {size of formula data below}
     gChanCnt       :integer;                  {# of component channels}
     gDataCnt       :integer;                  {# of values per channel}
     gDataWidth     :integer;                  {size of data in tables}
     gFormulaData   :array [0..gFormula size]
                     of byte;                  {data for gamma calculation
                                                formula}
     gData          :array [0..gData Cnt]
                     of byte;                  {gamma correction look-up tables}
                    end;
```

In this data structure, the `gVersion` field represents the gamma table format version, which is 0 for all current video cards. The `gType` field holds the `drHWId` value for this video card to identify the card for which this table was measured. This means that even if two different cards have the same CLUT response curve, they cannot share the same gamma table. When the value in the `gType` field is 0, the card should respond by examining the other fields in the table. The `gFormulaSize` field defines the number of bytes occupied by the `gFormulaData` field.

The Apple video cards currently used in Macintosh computers perform gamma correction by modifying the value loaded into the CLUT by the `SetEntries` control call to approximate a linear response on the video display. The gamma correction acts as a final look-up data table that translates the requested color into the closest available linearized level. These gamma table values are determined empirically by measuring the output of a calibrated display. The frame buffer of the Macintosh II Video Card uses a single correction table for all three channels and performs no calculations on the incoming color other than a simple look-up. The card remembers the specific monitor configuration at the beginning of the `gFormulaData` field, allowing it to identify and use only the gamma tables developed for the attached monitor.

The `gChanCnt` field is the number of look-up tables in the `gData` field. The R, G, and B tables follow each other, respectively, at the end of the structure if there is more than one channel of gamma correction data. The `gDataCnt` field gives the number of discrete look-up values included in each of the channel's correction tables.

The `gDataWidth` field describes the number of significant bits of information available in each entry in a channel's correction table. Since it is rare to have devices with more than 8 bits of CLUT resolution, virtually all devices pack their correction data into bytes.

The last field in the gamma table data structure, `gData`, represents the actual correction data. If more than one channel's information is present, a block of information for each channel appears in red, green, and, finally, blue channel order. Apple's video driver includes only one table that is applied to all three output channels.

In addition to the gamma table data structure, there is a standard resource format (resource type = `'gama'`) for gamma table resources. Like many other resource templates, the gamma structure is an image of the RAM form stored in resource format.

Using gamma correction

The video driver is responsible for applying gamma correction. First, the `Open` routine sets the default gamma table from the card's gamma directory. An `_InitGraf` call then causes the `'scrn'` screen configuration resource to be read from the System file. This resource is described with the Resource Manager information in *Inside Macintosh*. The resource includes information about the size and orientation of the different monitors configured into the system, including their last video mode (pixel size), color table, and gamma table. If no `'gama'` resource ID is specified, or if the specified ID is not present, a default gamma table, `'gama'` = 0, is loaded from the System file and used as the table for the Macintosh II Video Card. If the specified resource is found, the system loads the resource and issues a control call to the driver to make this the current gamma table.

The standard video driver includes two routines, the `SetGamma` control routine, which sets the gamma table, and the `GetGamma` status routine, which returns the pointer to the current gamma table. The `SetGamma` routine (`csCode` = `4`) and the `GetGamma` routine (`csCode` = `8`) were defined earlier in this chapter in the sections "Control Routines" and "Status Routines," respectively.

Video driver example

An example of a possible video driver is provided in Appendix C. The sample code is written in Macintosh Programmer's Workshop assembly language.

Summary

This section summarizes the video driver data structures, the slot interrupt queue routines, and the advanced control and status routines. It also gives guidelines for using assembly-language data structures, and installing and removing interrupt queue routines.

Data types

```
TYPE
VDParamBlockPtr = ^VDParamBlock;
VDParamBlock = RECORD
              qLink: QElemPtr;            {standard I/O param block}
              qType: INTEGER;
              ioTrap: INTEGER;
              ioCmdAddr: Ptr;
              ioCompletion: ProcPtr;
              ioResult: OSErr;
              ioNamePtr: StringPtr;
              ioVRefNum: INTEGER;
              ioRefNum: INTEGER;
              csCode: INTEGER;            {video driver specifics}
              csParam: Ptr;
              END;

VDEntRecPtr = ^VDEntryRecord;
VDEntryRecord = RECORD
              csTable: Ptr;               {pointer to color table}
              csStart: INTEGER;           {start entry number}
              csCount: INTEGER;           {count number}
              END;

VDGamRecPtr = ^VDGammaRecord
VDGammaRecord = RECORD
              csGTable: Ptr;              {pointer to gamma table}
              END;
```

```
VDPgInfoPtr = ^VDPgInfo;
VDPgInfo = RECORD
              csMode: INTEGER;            {mode within device}
              csData: LONGINT;            {data supplied by driver}
              csPage: INTEGER;            {page to switch in}
              csBaseAddr: Ptr;            {base address of page}
               END;

VDFlagPtr = ^VDFlagRec;
VDFlagRec = RECORD
              flag: SignedByte;           {used in various ways}
              END;

VDDefModePtr = ^VDDefModeRec;
VDDefModeRec = RECORD
              spID: SignedByte;           {spID}
              END;
```

Interrupt queue routines

```
FUNCTION SIntInstall(sIntQElemPtr: SQElemPtr; theSlot: INTEGER): OsErr;
FUNCTION SIntRemove(sIntQElemPtr: SQElemPtr; theSlot: INTEGER): OsErr;
```

Advanced control routines

Code	Name	Param	Effect
0	`Reset`	`VDPgInfoPtr`	Resets card to startup state
1	`KillIO`	None	Stops current and pending I/O
2	`SetMode`	`VDPgInfoPtr`	Changes card's video mode (pixel depth)
3	`SetEntries`	`VDEntRecPtr`	Changes card's color table (if any)
4	`SetGamma`†	`VDGamRecPtr`	Sets a gamma table
5	`GrayScreen`†	`VDPgInfoPtr`	Fills video page with gray
6	`SetGray`†	`VDFlagPtr`	Flags whether luminance mapping is on/off
7	`SetInterrupt`	`VDFlagPtr`	Enables/disables the interrupt handler
8	`DirectSetEntries`†	`VDEntRecPtr`	Changes color table (direct device only)
9	`SetDefaultMode`†	`VDDefModePtr`	Sets `spID` of default configuration in slot PRAM

† Denotes "optional."

Status routines

Code	Name	Param	Effect
0 and 1	N/A	N/A	Not implemented in video drivers
2	`GetMode`	`VDPgInfoPtr`	Returns mode, page, and base address
3	`GetEntries`	`VDEntRecPtr`	Returns color table entries
4	`GetPages`	`VDPgInfoPtr`	Returns number of pages in mode
5	`GetBaseAddr`	`VDPgInfoPtr`	Returns base address of page
6	`GetGray`	`VDFlagPtr`	Returns whether luminance mapping is on/off
7	`GetInterrupt`	`VDFlagPtr`	Returns state of interrupt handler
8	`GetGamma`†	`VDGamRecPtr`	Returns a pointer to set gamma table
9	`GetDefaultMode`†	`VDDefModePtr`	Returns `spID` of last set default configuration

† Denotes "optional."

Assembly-language information

Data structures

```
; use with Set/GetEntries

csFirst        EQU    0                    ; [word] first color table entry
csCount        EQU    csFirst+2            ; [word] number of entries to set
csTable        EQU    csCount+2            ; [long] pointer to color table
                                           ; entry = value, r, g, b : INTEGER

; use with control calls where csCode = 0, 2, 5, or 6
; and with status calls where csCode = 2, 4, 5, or 6

csMode EQU     0                           ; [word] mode within device
csData EQU     csMode+2                    ; [long] data supplied by driver
csPage EQU     csData+4                    ; [word] page to switch in
csBaseAddr     EQU    csPage+2             ; [long] base address of page

; use with Set/GetGamma

csGTable       EQU    0                    ; [long] pointer to gamma table
```

Interrupt queue routines

```
; to install a new queue element

       LEA            PollRoutine,A1        ;get routine address
       MOVE.L         A1,SQAddr(A0)         ;set address
       MOVE.W         Prio,SQPrio(A0)       ;set priority
       MOVE.L         A1Parm,SQParm(A0)     ;save A1 parameter
       MOVE.W         Slot,D0               ;set slot number
       _SIntInstall                         ;do installation

; to remove a queue element

       LEA            MySQEl,A0             ;pointer to queue element
       _SIntRemove                          ;remove it
```

Chapter 10 **NuBus Design Examples**

This chapter contains performance-proven examples of design that you can use to implement the NuBus interface in Macintosh computers.

◆ *Note:* The examples in this chapter were developed before the NuBus '90 specification was written. Therefore, they do not make use of the latest NuBus '90 features or signal lines. The design, however, will work on the entire line of Macintosh computers that offer a NuBus expansion interface.

NuBus Test Card

The NuBus Test Card (NTC) is an example of a complete master/slave NuBus slot card. In use, this card allows the Macintosh computer's central processor (or other NuBus master card) to test the functionality of the NuBus slave and master response logic. It provides an example of the type of logic necessary to implement a NuBus master card.

This description is to assist a hardware engineer who wants to see how a typical NuBus card is designed. No motivation for the design choices is given; it is intended as a description of an existing design. You should already be familiar with NuBus, PALs, and so forth.

Overview of operation

The NTC in slave mode is addressed by the microprocessor on the main logic board (or any bus master) and properly written to, so that the three NTC registers are set up with valid information. The microprocessor next addresses one of the registers to seek bus mastership; the NTC waits a programmed number of clock cycles and then arbitrates to become bus master. When it becomes bus master, the NTC accomplishes the read or write to an address that was stored in the NTC Address register.

Programming model

This section describes how the NTC looks to a programmer.

The NTC provides three registers: Address, Data, and Master. The three registers can be accessed by addressing the NTC as a *slave*. The first two registers, Address and Data, can be read from and written to; they support only NuBus word (32 bit) operations. Both of these registers can be used to test the basic data paths of the bus. However, these registers are primarily intended to supply the address and data that will be used during the NTC's master transaction when the NTC becomes bus *master.*

The 12-bit Master register is write only. When the Master register is written to, the NTC, after a programmed delay, initiates a transaction in which it becomes the bus master. The bits of the value written to the Master register are interpreted as shown in Table 10-1.

Bits 11 and 10 contain the /TM1 and /TM0 values that (along with address bits /AD1 and /AD0) define the transfer mode of the master transaction (see "Data-Transfer Specifications" in Chapter 3). Bits 7 through 0 contain the programmed time delay (in one's-complement form).

■ **Table 10-1** Master register interpretation

Bit	Assigned meaning
D11	/TM1 value (1 means /TM1 is asserted [low]), the read/write indicator
D10	/TM0 value (1 means /TM0 is asserted [low]), the data item length indicator
D9	Lock bit (1 means execute a locked transaction)
D8	0 (zero)
D7–D0	A one's-complement Delay value

After the execution of the write to the Master register, the master cycle is delayed by the number of clock periods specified by Delay. Delay is the value in the least significant 8 bits of the Master register; that value is incremented to $FF before the NTC becomes bus master and initiates a transaction.

The register addresses are given in Table 10-2; *s* is the number of the slot into which the card is inserted.

■ **Table 10-2** Register addresses

Address	Name
$Fss0 0000	Address register
$Fss4 0000	Data register
$Fss8 0000	Master (write-only) register

Byte swapping and the NTC

Byte swapping is necessary when interacting with the NTC because of the design of the NTC, the reordering of bytes when the computer transfers data across the NuBus, and the byte ordering of the NuBus.

As noted in Chapter 7, the NuBus interface performs a byte swapping of data values (see Figure 7-2 and the bus interface logic in Figures 1-1 through 1-7). For example, the byte containing bits D31–D24 of the microprocessor (referred to as byte 0) is swapped so that the byte is transferred to NuBus byte lane 0 (/AD7–/AD0). This preserves byte address consistency between cards on the NuBus. Every NuBus interface must be designed so that its byte 0 is placed on NuBus byte lane 0, byte 1 on byte lane 1, and so forth. If you transfer a microprocessor word of $0011 2233 to the NuBus, then, on the NuBus, it will appear as $3322 1100 (the bytes are displayed as most significant byte [msb] to least significant byte [lsb] in left-to-right order).

As can be seen on the schematic for the NTC (Foldout 7 in the back of the book), the Address and Data registers are connected so that a byte written to a given NuBus byte lane will be placed back on the same byte lane when these registers are read from as a slave or when driven to as a master. That is, there is no byte swapping performed by the NTC itself.

This design of the NTC has ramifications on how the values are written to its registers. For example, an Address register value must be byte swapped when written from the microprocessor. If we want the NTC to make a transaction to $1122 3344 (in NuBus format), we must write the data so that the msb of the Address register contains the $11 byte; this means that NuBus byte lane 3 must contain the $11. However, because NuBus byte lane 3 is driven by byte 3 from the microprocessor, the value we write must have the $11 in the lsb of the microprocessor value (where byte 3 belongs). Following this logic for the rest of the bytes, it should be apparent that the appropriate value to be written by the microprocessor to the NTC Address register is $4433 2211.

The same byte swapping must be done to values that are written to the Data register in preparation for a NuBus write by the NTC. Remember, however, that data values that are written to or read from the main logic board (for example, RAM) are byte swapped by the bus interface logic as the transaction is made. Thus, data values that are destined for (or read from) RAM will not look byte swapped. For example, suppose that we set up the NTC to read a RAM location that contains $1234 5678 (microprocessor form). When we read the Data register after the transaction is completed, we read $1234 5678. The reason is that when the NTC did the read, the bus interface circuits placed the data onto NuBus as $7856 3412 (due to the byte swapping of the bus interface). Then, when we read the Data register, the value is byte swapped by the bus interface circuits (again) so that the microprocessor sees the value as $1234 5678. If we wanted a *NuBus* value of $1234 5678, then the appropriate microprocessor value would be $7856 3412.

△ **Important** In terms of the Macintosh family, a value may be specified from the perspective of the computer's microprocessor or of the NuBus interface. Values viewed from the NuBus interface need to be byte swapped; values viewed from the microprocessor do not. △

Programming the NTC

In the following discussions, values for various NuBus fields are specified. In all cases, the values are the logical values; remember that these are the complements of the NuBus signals. For example, if /TM1 is a 1, then that implies that /TM1 (the NuBus signal) will be *low.* Also, all references to data width will be in NuBus terms—that is, NuBus word (32 bit), halfword, and byte.

The following two steps are necessary for the NTC to perform a master cycle:

1. The Address register is set up with the desired master transaction's address; a byte-swapped value must be written to the Address register. The lower 2 bits of the Address register become part of the NuBus transfer mode; the values of these 2 bits must be modified to correspond to the desired transfer mode encoding, not what the microprocessor program would use for the equivalent access.

   If the master transaction is to be a *write,* then you must write data to the Data register that will be transmitted when the NTC becomes bus master.
2. The proper /TM1–/TM0, Lock, and Delay values are written to the Master register. The NTC waits for the number of clock periods specified by the Delay value, and then makes the master transaction.

Included in this section are two examples of setting up the NTC to execute master transactions. In these examples, references to the NuBus transfer mode will be given in the 4-bit form </TM1,/TM0,/AD1,/AD0>, where the bit values represent the corresponding NuBus signal level—H for high and L for low. Values of the Master register bits for /AD1–/AD0 and /TM1–/TM0 are the logical values (0 and 1). Remember that a 0 written to a register will be placed on the NuBus as an H (and a 1 as an L).

Word read (Macintosh computer RAM)

Suppose that you wish to cause the NTC to perform a word read transaction to location $1234; this causes a read of the computer's RAM. The proper NuBus transfer mode for reading a NuBus word is <HHHH>, as shown in Table 3-1. Thus the values written to /TM1–/TM0 and /AD1–/AD0 are adjoined to form the 4-bit transfer mode code <0000>. Hence, the microprocessor writes the following values into the registers:

$3412 0000 into Address
$0000 00FF into Master

This causes the NTC to execute a word read (because /TM1–/TM0 and /AD1–/AD0 are all 0) from location 0 immediately (FF is the one's complement of 00, for a Delay value of zero clock periods).

Halfword 0 write

The proper transfer mode value is <LHHL>, from Table 3-1. Therefore, the value for </TM1,/TM0,/AD1,/AD0> is <1001>, and the registers are programmed to be loaded as follows:

$3512 00F9 into Address
$7856 xxxx into Data
$0000 08BF into Master (D11–D8 in Table 10-1; binary 1000 is 8 in hex; $BF is the one's complement of $40)

◆ *Note:* In the last nibble of $1234, $4 = 0100 in binary, so /AD1 = 0 and /AD0 = 0. But /AD1 and /AD0 must be changed to encode the least significant 2 bits of the transfer mode, so /AD0 is changed to a 1 and now 0101 = $5. Then $F900 1235 becomes $3512 00F9 when byte swapped. The address $F900 1234 on the NuBus is obtained by writing $3512 00F9 into the Address register.

Data is written to the Data register so that when the master transaction is performed, the data will be in the proper byte lanes. Halfword 0 data is contained in byte lanes 1 (msb) and 0 (lsb). Hence, you need to write the data from the microprocessor such that $56 (msb) is in byte 1 and $78 (msb) is in byte 0. The microprocessor must write the value $7856 xxxx.

Hardware organization

This section describes the hardware used to mechanize the NuBus Test Card. The schematic is shown in Foldout 7 at the end of the book. PAL equations are displayed in Appendix D.

The NTC consists of

- four NuBus address/data buffers (74ALS651's), U1–U4 (also called transceivers)
- eight octal latches (74ALS374's), U5–U12, which implement the Address and Data registers
- one 74F86, U14, and one 74F30, U15, which form an address comparator
- five PALs, U16–U20, which implement the control logic
- one ROM socket, U13, for the declaration ROM
- two 4-bit counters (74ALS161's), U23–U24, which implement the Delay counter
- one 74F04 inverter, U21
- one 74F02 NOR, U22

NuBus address/data buffers

The NuBus address or data buffers, U1–U4, are grouped into two parts. U1 can be independently driven onto the NuBus, while U2–U4 are latched into the NTC. This allows the addresses for the ROM to be held during a ROM read cycle without additional parts. For all other operations, all of the buffers are set for transferring data from or to the bus in unison.

Address and Data registers

The two sets of latches (U5–U8, U9–U12) form the Address and Data registers. They are latched during a write to the corresponding register and are enabled either upon a slave read to the register or during a master transaction.

Address comparison

U14 and U15 are wired so that the output of U15 is low when an address of $Fsxx xxxx is present on the /AD lines. This signal is used by the slave PAL to detect the start cycle to the card.

SLAVE PAL

The slave PAL (SLAVE PAL) is the state machine for slave accesses to the NTC. It also latches the state of /AD19–/AD18, which are used by other PALs.

ARB PAL

The arbitration PAL (ARB PAL) is responsible for performing the NuBus arbitration process. When /ARBCY is asserted, the /ID3–/ID0 value drives the /ARB3–/ARB0 lines. However, when /ARB detects that a higher-priority value is present on the /ARB3–/ARB0 lines, it removes drive from its lower-priority lines, following the NuBus rules. The GRANT signal is asserted when /ARB recognizes that its /ARB3–/ARB0 value is valid; GRANT is used by the master PAL to detect that the NTC has won ownership of the bus.

MASTER PAL

The master PAL (MASTER PAL) is responsible for controlling a master transaction on the bus. It idles until it detects that both the MASTER and MASTERD (delayed MASTER) input signals are true. It will then go through a state sequence to perform the transaction. The master PAL can execute two types of transactions: normal and locked. The state sequence is slightly different for each case. See the timing diagram in Figure 10-1 for the sequences of each. Note that the diagram shows the shortest slave response. In actual use, most accesses hold in the wait state (/DTACY asserted) while awaiting an /ACK for more than one cycle.

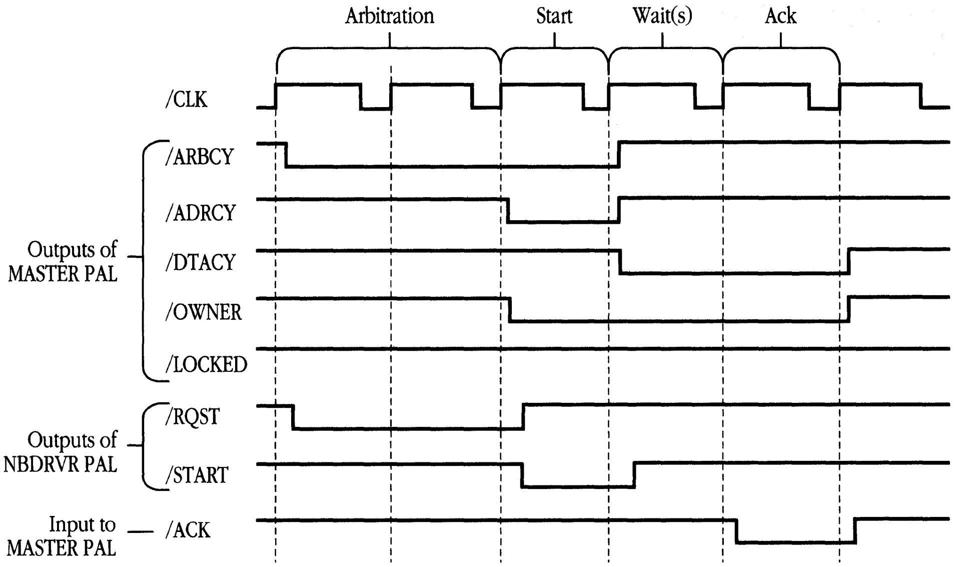
MISC PAL

The miscellaneous PAL (MISC PAL) is used to decode the state machine signals and drive on-card devices. The outputs control the gating of the 651's, 374's, and so forth.

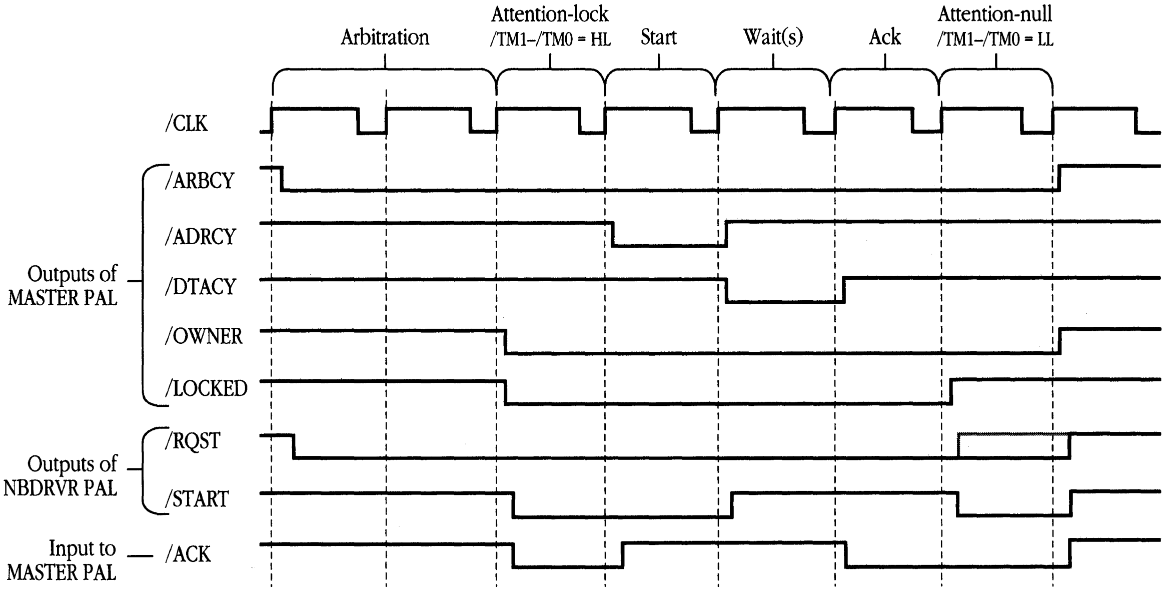
NBDRVR PAL

The NuBus driver PAL (NBDRVR PAL) is responsible for driving all NuBus signals. As in the miscellaneous PAL, NBDRVR decodes the state machine signals to determine the timing for these signals.

■ **Figure 10-1** Master transaction timing, normal and locked



Master transaction (normal)



Master transaction (locked)

Slave operation

During a slave access by another master, the operation of the NTC is determined by the slave, miscellaneous, and NuBus driver PALs. The slave PAL determines that an access to the NTC is being made (by looking at the slot decode, /START, and /ACK) and performs timing. The miscellaneous PAL determines whether to clock (/ACLK or /DCLK) or output enable (/AOE or /DOE) the 374's, enable the appropriate 651 direction, and so forth, based upon the inputs from the slave PAL.

When the slave PAL detects that the Master register is being written to, it will finish the slave access and set its MASTER output signal. During the data cycle of the Master register write, the slave PAL latches the values of D11–D10 and causes the values of D7–D0 to be latched into the 161 counters. During the subsequent master transaction, the slave PAL will not respond until the /MSTDN signal is asserted.

Master operation

A master transaction is begun when the slave PAL sets the MASTER signal. After the 161's have counted up to $FF, the master PAL begins the master state sequence.

After arbitration, the master PAL does its start cycle and waits for the acknowledge cycle. When /ACK is detected, /MSTDN is signaled; this causes the slave PAL to start looking for new slave transactions to the NTC.

△ **Important** This design violates the letter of the law of NuBus in one regard; however, this violation causes no problem in a real system. The violation occurs at the end of a locked transaction. The /RQST signal is held asserted during the final attention-null cycle; it should be released during that cycle. No problem exists, because either the NTC is the last request (/RQST) or it is not. If it is the last, then the only effect is that new requestors must wait an additional clock cycle. If it is not the last, then /RQST would stay asserted anyway. In either case, the proper operation of the bus ensues. △

SCSI-NuBus Test Card

The SCSI-NuBus Test Card is an example of how a simple, 8-bit I/O chip may be supported over NuBus. This card allows the test of declaration ROM images, in particular, the Slot Manager. The card allows an image of a bootstrap program (contained in the card's declaration ROM) to boot the Macintosh Operating System from an attached SCSI drive. In addition, the card provides a small RAM, which is accessible in super slot space for the testing of 32-bit address mode switching.

The ROM is really a RAM that you can write to at the assigned ROM address space. The RAM chip may be replaced with a real ROM when desired.

Software overview

The software model of this card is essentially the same as that of the SCSI chip on the main logic board, except that it is accessed via NuBus. The address offsets of the registers and pseudo-DMA are the same as on a Macintosh SE or Macintosh Plus.

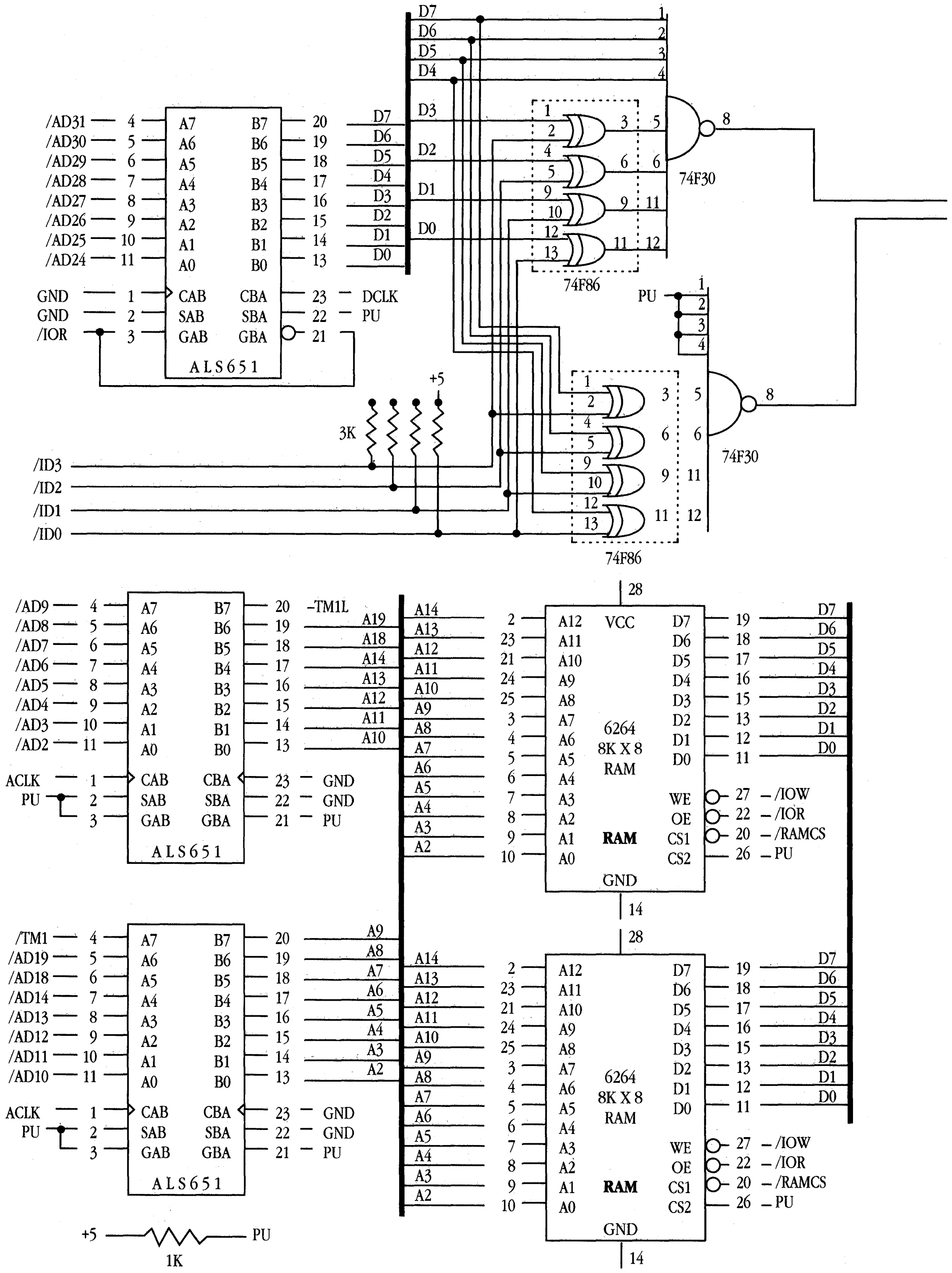
The SCSI chip can generate NuBus interrupts (via /NMRQ) from both IRQ and DRQ; this interrupt can be disabled.

The declaration ROM is accessed at the top of the 1 MB address space. The SCSI chip is accessed at the bottom of the space. The 8 KB of RAM is accessible only as a super slot. Note that all of the devices are connected to byte lane 3 (bits /AD31–/AD24) of NuBus. They are thus addressed from the microprocessor as bytes at addresses with the least significant 2 bits equal to 3 (/AD1 = /AD0 = 1, low). See Table 3-1, Figure 7-1, and the NuBus Test Card examples earlier in this chapter.

Hardware overview

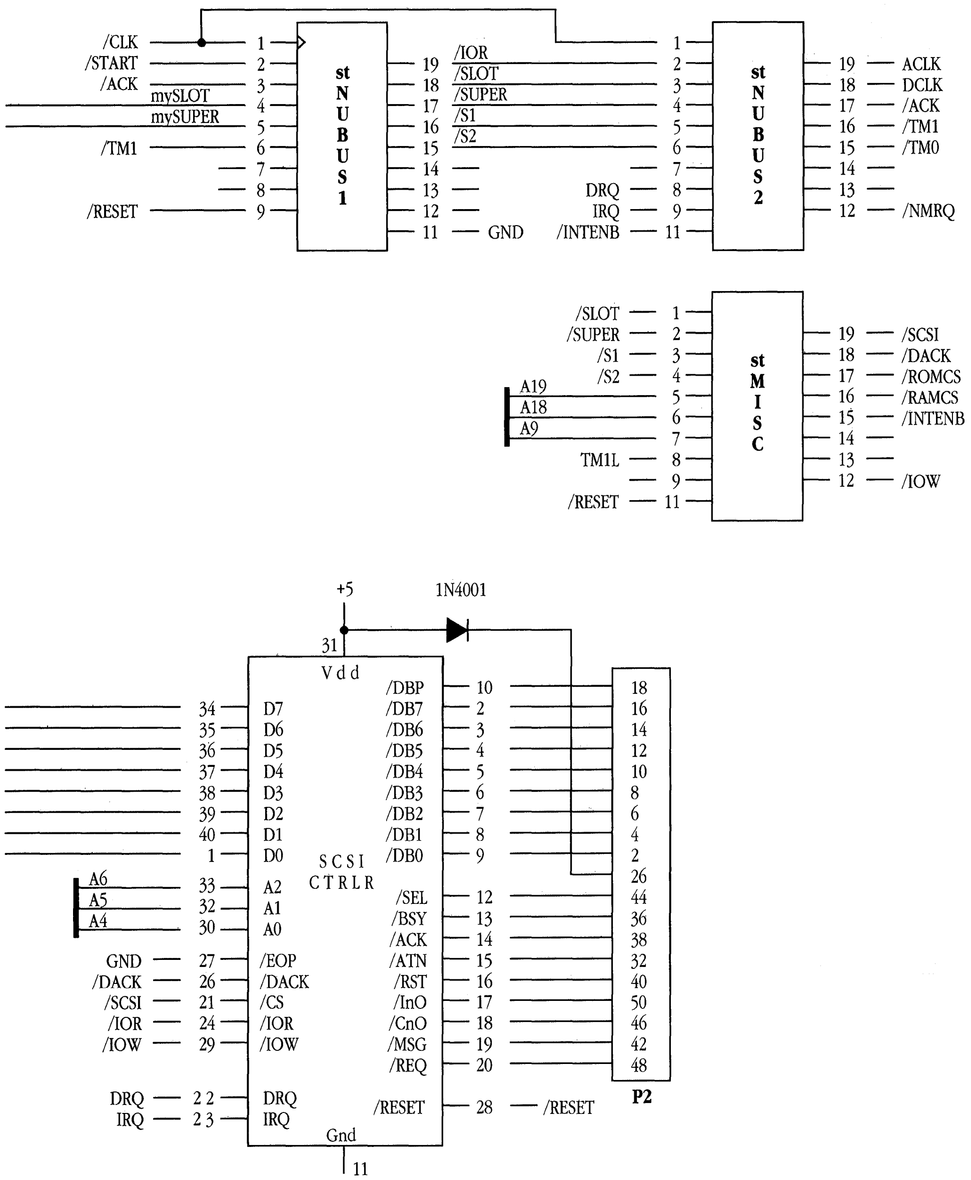
This section describes the hardware components and how they function. Figure 10-2 is an electrical schematic of the SCSI-NuBus Test Card; Figure 10-3 is the timing diagram. The PAL equations are in Appendix E.

■ **Figure 10-2** Schematic of SCSI-NuBus Test Card



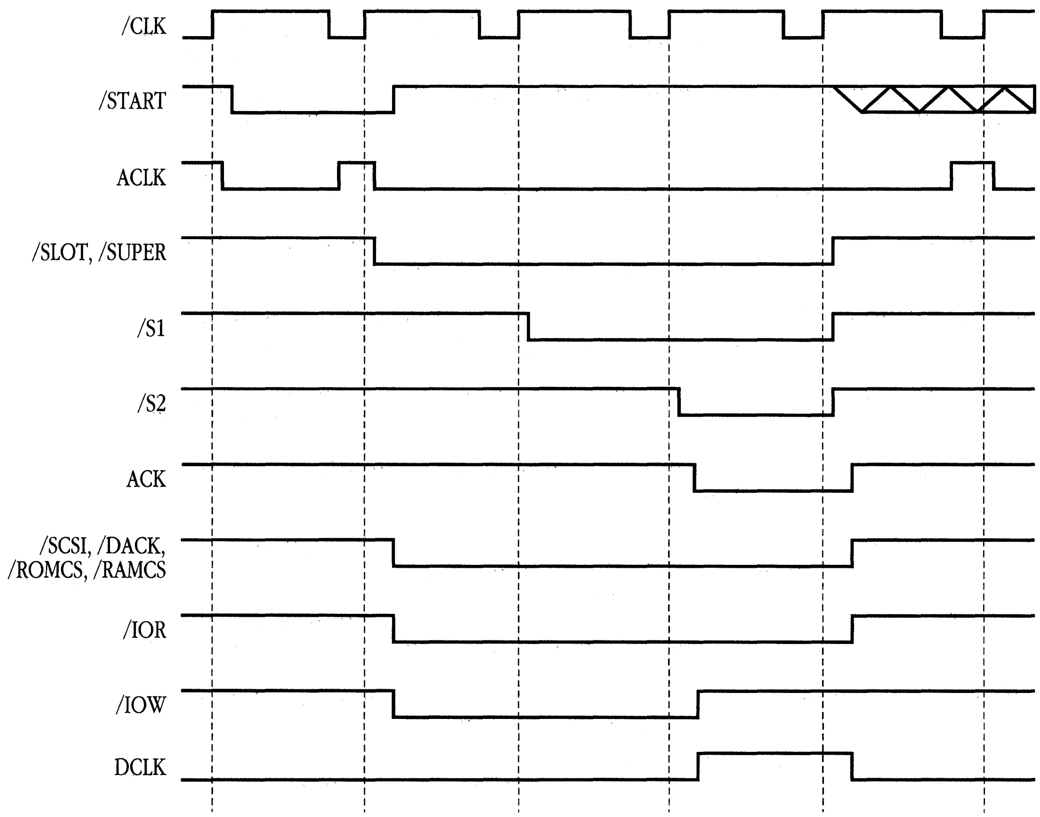
(continued)

■ **Figure 10-2** Schematic of SCSI-NuBus Test Card (continued)



Note: All IC terminals and lines labeled Gnd or GND are connected to power ground.

■ **Figure 10-3** SCSI-NuBus timing diagram



NuBus transceivers (ALS651's)

Three 74ALS651's are used to implement the NuBus transceiver function.

One of them is the data transceiver; it connects to byte lane 3 (bits /AD31–/AD24) and serves to transmit and receive the bytewide data over NuBus. During idle states, the data transceiver is also monitoring the bus to feed data into the slot decode logic.

Two 74ALS651's are used to latch addresses (/AD14–/AD2, /AD18, /AD19) and the write/read signal (/TM1) for the SCSI, ROM, and RAM accesses. These chips are clocked by a signal from stNUBUS2 every falling edge of /CLK until stNUBUS1 detects an access to the card. They then hold onto the low-order address bits that were present during the transaction's start cycle.

Slot Decode (F86/F30)

The Slot Decode card uses a combination of a 74F86 and a 74F30 to perform slot decoding. Two sets are used, one for the standard slot space decode ($Fsxx xxxx) and the second for the super slot access decode ($sxxx xxxx).

NuBus state machine (stNUBUS1 PAL)

The NuBus state machine PAL (16R8B) performs the basic NuBus timing for the card. When either mySLOT or mySUPER is detected during a start cycle, the PAL generates /SLOT or /SUPER and starts a 2-bit counter (/S2, /S1), which is used by s/TMISC. The value of /TM1 during the start cycle is latched to form the /IOR signal, the assertion of which indicates a read.

NuBus signal generator (stNUBUS2 PAL)

The NuBus signal generator PAL (16L8B) decodes the state of /SLOT, /SUPER, and /S2 to generate the acknowledge cycle and control the latching of the 651's.

The stNUBUS2 PAL is also used to generate the open-collector /NMRQ signal for presentation of interrupts to the main logic board.

Decode and timing (stMISC PAL)

The decode and timing PAL (16L8B) generates the basic I/O strobes to the SCSI, ROM, and RAM. It uses the /SLOT and /SUPER signals in addition to the latched address bits to perform the decode.

The INTENB signal is a latch that controls the generation of /NMRQ. It is set by addressing $Fsx 820x; it may be cleared by addressing $Fsx 800x.

SCSI chip (NCR5380)

The SCSI chip is identical to that used in the Macintosh Plus. It connects to a SCSI bus via the connector P2, which also supplies the TRMPWR signal for SCSI termination.

Pseudo-ROM

The ROM of this card was designed to allow software designers quick update capability. It is really an 8 KB × 8 RAM that can be written to using the ROM address space. However, a real 8 KB × 8 ROM may be inserted instead.

RAM

The RAM chip is an 8 KB × 8 RAM that is accessible only by addressing super slot space.

PAL descriptions

The source code of the three PALs is in Appendix E. Refer to these PAL equations, along with the timing diagram and schematic (Figures 10-2 and 10-3), for a more detailed understanding of how the card works.

A simple disk controller

This section describes the electrical and interface characteristics of a slave-only disk controller card that allows a Macintosh computer to communicate with a generic disk drive through the NuBus.

The disk controller card plugs into any NuBus slot on the main logic board and connects to a floppy disk drive located outside the computer. The disk controller card consists of a disk controller IC and a disk interface IC, a sector buffer RAM, a declaration ROM, various address and data buffers, and three 24-pin PALs. All controlling firmware exists in the computer. The controller is memory mapped into a single NuBus slot space.

System configuration

The controller package consists of a disk controller card, a cable running from controller to disk drive, and a floppy disk drive. The disk controller card connects the disk drive to the computer's central processor through one of the slots on the main logic board. One end of the cable connects to the controller card, and the two connectors on the other end of the cable connect to the disk drive.

Controller card block diagram

The controller card is made up of the following parts, shown in Figure 10-4:

Address/data bus transceivers: The address/data bus transceivers (74LS640-1's) buffer the internal address/data bus of the controller from the NuBus address/data bus.

Address counters: The address counters (74LS169's) latch the RAM/ROM address from the NuBus during RAM/ROM reads or writes and count down the RAM address during DMA transfers to or from the disk.

RAM: The RAM is the 2048 × 8 sector buffer RAM. Data to be transferred to or from the disk is placed here by the processor before disk transfers are initiated.

ROM: The ROM is the NuBus declaration ROM. The NuBus Slot Manager accesses this ROM on power-up to determine the controller's type and modes of access.

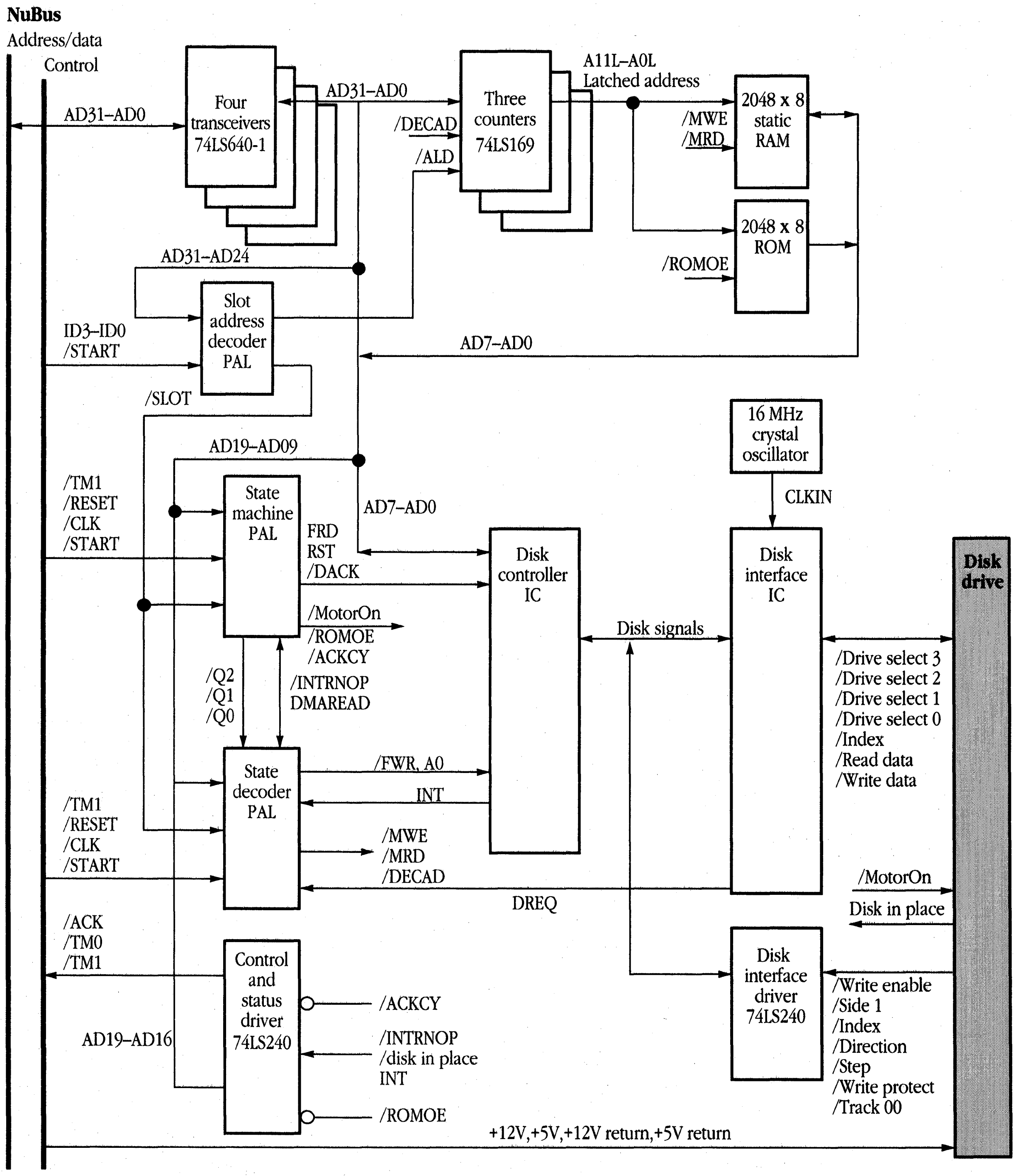
Slot address decoder PAL: The slot address decoder PAL (PAL20L10) determines if the controller's slot address is selected. It uses the signal /START and address decoding to compare if the upper nibble of the address is an $F and if the address lines A27–A24 and D3–D0 compare with the hard-wired slot ID address.

State machine PAL: The state machine PAL (PAL20X10) generates the timing for programmed I/O and internal DMA transfers on the controller.

State decoder PAL: The state number is decoded by the state decoder PAL to produce control signals needed by the various parts of the controller.

Control/status driver: The control driver places the signals /ACK, /TM0, and /TM1 on the NuBus at the end of a NuBus access of the controller. The status driver allows the following signals to be read by the processor: disk controller interrupt, internal operation pending, and disk in place.

■ **Figure 10-4** Floppy disk controller block diagram



Floppy disk controller IC: The floppy disk controller LSI chip contains the circuitry necessary to communicate with the generic disk drive. Coupled with the companion disk interface IC chip, it handles all operations with the drive, including reading and writing data, formatting, seeking, sensing drive status, and recalibrating.

Floppy disk interface IC: The floppy disk interface chip provides drive and timing support to the disk controller IC. It contains write precompensation and phase-locked loop circuitry.

Disk interface driver: The disk interface driver buffers and provides current drive for several signals coming from and going to the disk drive. It also is used as a multiplexer for four signals: FLT/TR0, WP/TS, FR/STP, and LCT/DIR.

16 MHz crystal clock oscillator: The crystal clock oscillator provides a 16 MHz clock to the disk interface IC for use in the drive interface.

Floppy disk controller logic

The disk interface is provided by the disk controller IC, the disk interface IC, and two 74LS240 drivers. The disk controller IC is the controlling chip and communicates with the disk interface IC. Details of this logic are not directly relevant to design of NuBus interfaces and so are not given here.

NuBus interface logic

The controller connects to NuBus via several drivers and PALs. The address/data bus is tied to four 74LS640-1 transceivers that invert each bit. Control signals such as /START, the slot identification bits /ID3–/ID0, and the mode bits /TM1–/TM0 are used to time data transfers to and from the NuBus. Status information is passed to the NuBus along with the control signal /ACK by the status driver (74LS240). DMA operations are controlled by the state machine and state machine decoder PALs.

Key RAM access signals are described in Table 10-3.

■ **Table 10-3** RAM access signals

Signal name	Signal description
A0	Disk controller IC register select: 0 selects main status register; 1 selects data register
/ACKCY	Gates /ACK and /TM1–/TM0
/ALD	Used to load the RAM/ROM address into the address counters; gates the clock signal into the synchronous counters
/DACK	Acknowledges the DMA cycle requested
/DECAD	Enables the DMA address counters to decrement by one memory location
/DMAREAD	Indicates a DMA read operation when asserted
/DREQ	Requests DMA cycle from disk controller IC or disk interface IC
/FRD	Enables disk controller IC read enable
/FWR	Enables disk controller IC write enable
/INTRNOP	When asserted, indicates internal DMA operation in process
/MRD	Enables RAM memory read output
/MWE	Enables RAM memory write
/SLOT	Signals that a NuBus cycle to the controller is active
SR	Direction signal to bidirectional driver on the address/data bus: 0 means write to NuBus, 1 means read from NuBus

Programmed I/O operations

Control and status information is passed to and from the controller using programmed I/O (PIO) operations. PIO transfers include RAM and disk controller IC reads and writes, and ROM reads. The /MotorOn and /RESET signals are asserted and deasserted using PIO operations. Refer to Figure 10-4.

A typical PIO transfer begins with the assertion of the signal /START. The slot address is valid during the time /START is asserted and is recognized by the slot decode PAL. It asserts the signals /SLOT and /ALD. The /SLOT signal indicates that the NuBus cycle is currently active. The /ALD signal is used as a clock enable signal for loading the RAM or ROM address into the counters. The /ALD signal is also used as a clock enable to latch /TM1 and address bits A19/D11, A18/D10, and A17/D9. These are later used to assert the signals /FRD, /FWR, /MWE, /MRD, SR, /ROMOE, and A0. The state machine, recognizing /SLOT, begins sequencing through a NuBus cycle, going to states 1, 3, and then 2. In state 2 it asserts /ACKCY, which in turn enables the status driver to assert /TM1–/TM0 and /ACK. The signals /FRD, /FWR, /MWE, /MRD, SR, /ROMOE, and A0 are asserted or deasserted according to the address on the address/data bus during /START and the state number.

The signals /FRD, /FWR, and A0 transfer data to and from the disk controller IC.

RAM accesses are controlled by /MWE and /MRD. The ROM is read when /ROMOE is active. The signal SR is used to control the direction of the 74LS640 transceivers.

On-card DMA operations

Direct memory access (DMA) operations transfer data to and from the sector buffer RAM. On-card DMA operations are not done through the NuBus because this card is a slave only.

The state machine is placed in internal DMA mode by writing to an address in the range $FssC 0000 through $FssF FFFF. See the next section, "Memory Map and the Declaration ROM," for the rationale behind the *ss* in these addresses. DMA operations from the disk to RAM require that the last command word to the disk controller IC be written to a location in the range from $FssC 0000 through $FssD FFFF.

DMA operations from RAM to the disk require that the last command word to the disk controller IC be written to a location in the range from $FssE 0000 through $FssF FFFF.

After a DMA operation has been requested, transfers to or from the disk are then initiated and controlled internally. After an operation is complete, the controller interrupts the processor. The address bits A13–A2 are the beginning RAM memory location that the DMA operation uses. This address is decremented until it reaches 0 and terminates the DMA operation.

An attempt to read or write to any address in the controller's address range during a DMA operation is ignored, although the NuBus cycle is terminated with normal status.

When a DMA operation is requested, the signal /INTRNOP is asserted along with /DMAREAD if the operation is a DMA read. A /SLOT or a /DACK signal causes the state machine to begin sequencing. Because the /DACK signal holds off /SLOT, if both happen simultaneously, the DMA operation is first completed, and then the NuBus cycle is acknowledged.

The signal /DACK occurs on the first rising edge of /CLK after the signal DREQ is asserted, and is held until the DMA cycle is complete. The disk-controller-IC/disk-interface-IC pair initiates the DMA cycle by asserting DREQ.

Memory map and the declaration ROM

The controller's device select space ranges from $Fss0 0000 to $FssF FFFF and is divided into eight blocks. The designator *ss* is used to indicate the slot space where *s* is the slot number and ranges from $9 through $E in the Macintosh family.

Table 10-4 summarizes the address decodes.

■ **Table 10-4** Device select decode addresses

Address range	Device selected and action resulting
$Fss0 0000–$Fss1 FFFF	Read status information from disk controller
$Fss2 0000–$Fss3 FFFF	Read control information from or write control information to the disk controller
$Fss4 0000–$Fss5 FFFF	Begin internal DMA cycle reading data from disk
$Fss6 0000–$Fss7 FFFF	Begin internal DMA cycle writing data to disk
$Fss8 0000–$Fss9 FFFF	Enable RAM for reading or writing
$FssA 0000–$FssB FFFF	Reserved
$FssC 0000–$FssD FFFF	Turn drive motor on by writing; turn motor and controller's reset signal off by reading (interrupts are enabled when the motor is on)
$FssE 0000–$FssF FFFF	Access ROM by reading; turn controller's reset signal on by writing

It is through the data register that commands, data, and values in status registers 0–3 are passed. Any disk operation is initiated by passing the several commands required to the disk controller IC via this register. If a format, read data, read deleted data, write data, or write deleted data command is requested, the data or parameters required by the disk controller IC during its execution phase must have been previously loaded into the sector buffer RAM.

The final command code written to the disk controller IC is written via the DMA execute addressing space. The read track operation is not supported because the quantity of data transferred exceeds the sector buffer size. After the execute portion of an operation is completed, the disk controller IC may give back status information in status registers 0–3.

To read the status of the disk controller, an additional status register is provided. This register is accessed by a `MOVE.W` to the address space from $FssE 0000 through $FssF FFFF (ROM).

Chapter 11 **The Macintosh II Video Card**

This chapter describes the video card designed by Apple for use in the Macintosh II family of computers. The purpose of this information is to provide you with an overview of good video card design, but not with step-by-step instructions for actually implementing the design. It is assumed that you have already read the NuBus design guidelines in Chapters 2 through 10. Although the material presented in those chapters is pertinent to all types of NuBus expansion cards, it is particularly appropriate to the design of a video card.

The Macintosh II Video Card described in this chapter is no longer available. For more information about the current Apple video cards, contact APDA. Specifically, refer to the *Display Card Developer Notes for the Macintosh Display Cards 4•8, 8•24, and 8•24 GC,* APDA publication number M0857L/A.

◆ *Note:* The examples in this chapter were developed before the NuBus '90 specification was written. Therefore, they do not make use of the latest NuBus '90 features or signal lines.

Video card overview

The original Macintosh II Video Card and the Macintosh II High-Resolution Video Card are high-performance color video cards for use with the computers in the Macintosh II family. These cards provide variable-depth color graphics at up to 8 bits per pixel. The cards contain a color look-up table (CLUT) with a 16.8-million-color palette and an 8-bit digital-to-analog converter (DAC) for each of three channels (red, green, and blue).

The original video card has several important features, including

- display resolution of 640 × 480 pixels
- refresh rate of 67 Hz for reduced flicker
- up to 256 colors out of 16.8 million possible
- support for 1-, 2-, 4-, and 8-bit pixel modes
- frame buffer sizes of 256 KB and 512 KB, user upgradable
- plug-in-and-go operation—requires no special configuration of hardware or software

In addition to the above features, the high-resolution video card provides some features not found on the original card. The new features are

- full support for RS-170 video monitors
- support for multiple screen sizes
- ability to recognize different monitors at startup time and automatically configure itself appropriately
- full support for A/UX in the card's ROM

Unless specified, the information in the following sections pertains to both versions of the video card. Firmware support is provided by the card's declaration ROM. The declaration ROM contains a low-level card driver that performs all of the interface and hardware management functions for the video card. The declaration ROM is described later in this chapter. The firmware structure of the declaration ROM is described in more detail in Chapter 8.

Operating-system support, as provided by Color QuickDraw, the Color Manager, and the Slot Manager, is detailed in *Inside Macintosh.*

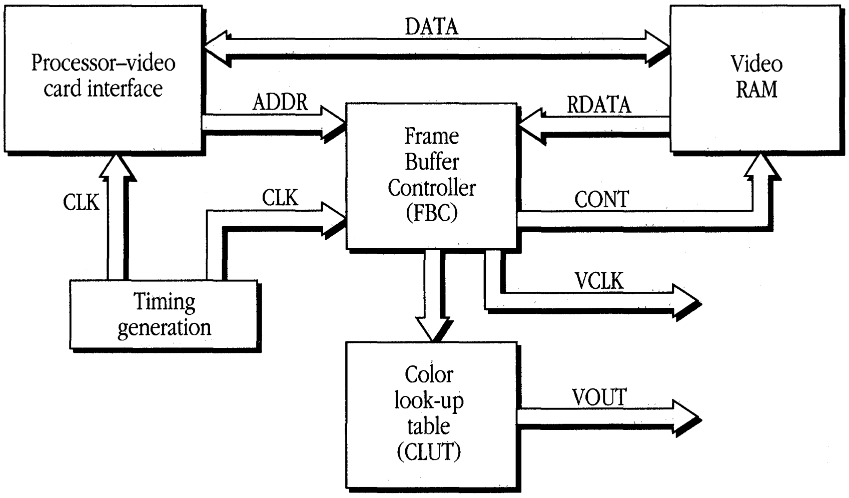
△ **Important** In addition to the 256 KB to 512 KB of video memory that QuickDraw manages, most of the video card features are subject to software control through several control addresses. These addresses are all located in the 16 MB slot space described in Chapter 7, "NuBus Card Memory Access." Since there is a difference between NuBus address allocation and the mapping of address space in Macintosh computers, you must be aware of the byte swapping that takes place on the main logic board of the computer. For more information on byte swapping, refer to the section "NuBus Bit and Byte Structure" in Chapter 7 and the section "Byte Swapping and the NTC" in Chapter 10. △

Functional operation

The video card controls the output of data to a video device through the use of the Frame Buffer Controller (FBC) and the color look-up table (CLUT). The declaration ROM provides the interface between the card hardware and application software running on the CPU.

Figure 11-1 is a block diagram of the video card. The following paragraphs briefly describe the function of each of the blocks shown in Figure 11-1.

■ **Figure 11-1** Video card block diagram


DATA
Processor–video card interface
Video RAM
ADDR
RDATA
Frame Buffer Controller (FBC)
CLK
CLK
CONT
VCLK
Timing generation
Color look-up table (CLUT)
VOUT

Processor–to–video card interface

The processor–to–video card interface is implemented by a combination of hardware and firmware. The hardware is the standard NuBus electrical interface, described in Chapters 1 through 7. The firmware is implemented in the declaration ROM, described later in this chapter and in Chapter 8, "NuBus Card Firmware."

Timing generation

The timing generation circuitry includes pixel clock oscillators that define the time for a single pixel. The latest version of the video card has two pixel clock oscillators, one for the Macintosh II–family monitor (30.667 MHz) and another for the Apple IIGS (RS-170) monitor (12.24 MHz). Only one of these clocks is active, as selected by firmware.

△ **Important** If the clock selected is not the right one for the type of monitor connected to the card, the display will not be readable. The card's firmware (Primary Init) stores information about the monitor so that software can't switch to the wrong clock. △

The timing generation circuitry generates timing signals for other devices on the video card, including

- the Frame Buffer Controller (FBC) interface signals
- the NuBus handshake and control signals
- other video card control signals

Frame Buffer Controller

The **Frame Buffer Controller (FBC)** is the most important single part of the card. It manages the video RAM, generates the video sync signals, and contains the NuBus interface circuitry. The FBC controls the transfer of data out of the serial port of the video RAM and into the CLUT/DAC, where the pixel values are converted into video display signals.

The FBC is a register-controlled CMOS gate array. The video card firmware controls the FBC by loading its set of control registers with the parameters stored in the declaration ROM. These registers are loaded during video card primary initialization and on certain video driver control requests. These operations are described later in this chapter under "Firmware Interfaces."

The FBC uses the parameters stored in the control registers to generate and control video data and timing signal output. Register contents determine video characteristics such as bit depth and timing. The registers are also used for other control functions such as selecting the appropriate pixel clock and reading the monitor sense line.

The various gated inputs on the FBC are used to execute RAM read/write and refresh operations. RAM operations are more fully explained in the next section, "Video RAM."

The control registers used by the FBC are mapped into the computer's main memory in the slot space assigned to the video card. The control address space is independent of the frame buffer data space.

△ **Important** Your applications should never access the hardware directly because the locations and functions of the registers may change (and also because the control registers won't be compatible with other manufacturers' cards). For this reason, the parameters stored in the FBC control registers are not documented in this book. To maintain product compatibility across a possible variety of Macintosh video cards, and to allow for any future changes to the hardware, you are strongly advised to always use software interfaces (driver routines) to control the operation of the video card. △

Video RAM

The video RAM makes up the frame buffer: the memory dedicated to storage of the pixel data for display. The frame buffer consists of two 256 KB banks of video RAM, Bank A and Bank B. Each bank of video RAM consists of eight ICs, each of which is a 64 KB × 4 RAM device with 150 ns access time. On a card with 256 KB of video RAM, only Bank A is populated; on 512 KB cards, video RAM chips are installed in both banks. The video card's firmware performs a test at startup time to determine the amount of video RAM installed.

The video RAM ICs are dual ported: in addition to the normal parallel port for reading and writing, each video RAM IC has a built-in shift register and separate serial port for video data. QuickDraw writes into the video RAM through the parallel port, and the FBC extracts the display data through the serial port. This separation of functions allows more than 95% of the video RAM's bandwidth to be available to the processor.

Of primary interest to you as a developer of a card or driver are NuBus operations to and from video RAM. Bus operations to RAM (transactions) are of two types:

- video RAM space writes and reads
- control space writes and reads

Figure 11-2 shows a timing diagram for a processor access to video RAM space, for writing and then reading. The typical sequence of functions is shown on the figure; also shown are the start and acknowledge cycles that characterize a transaction, as described in Chapter 2, "NuBus Overview," and Chapter 3, "NuBus Data Transfer."

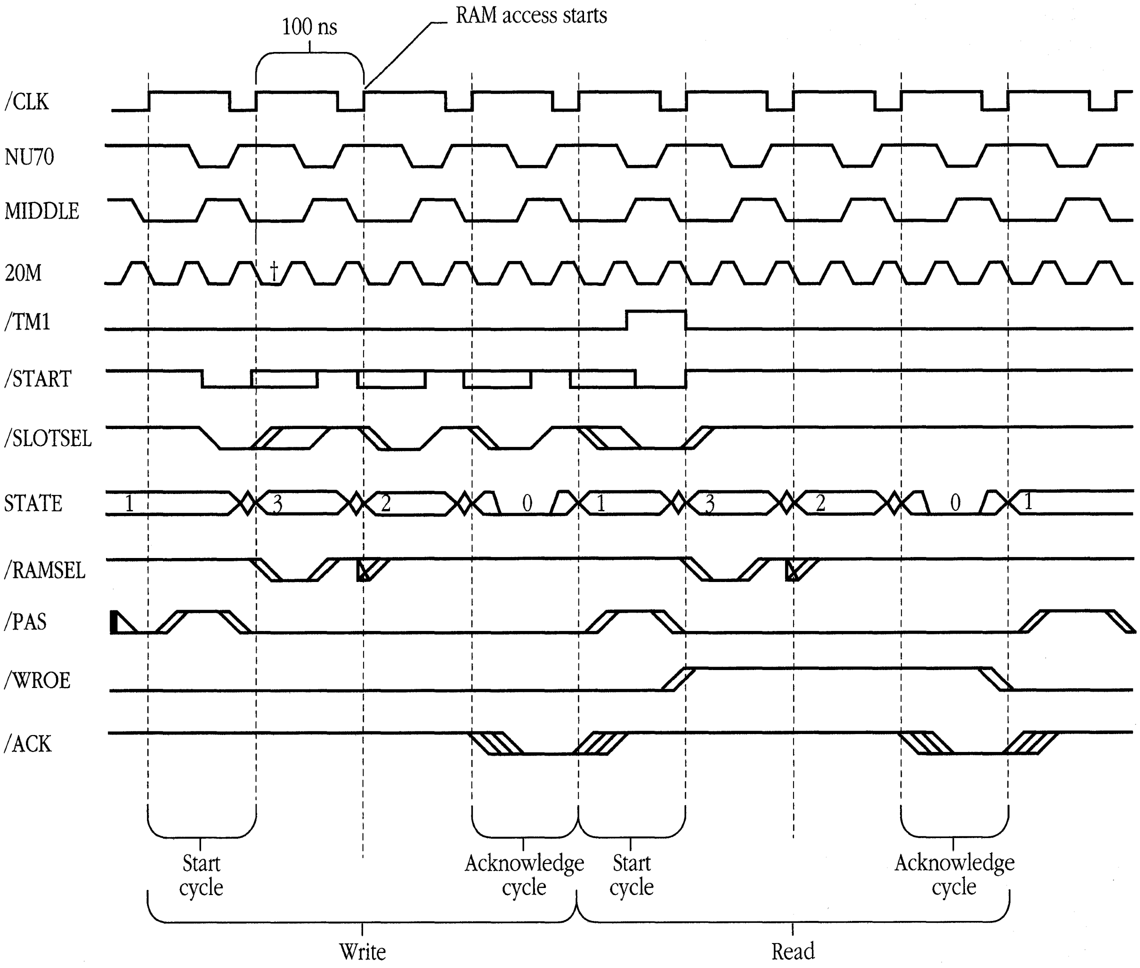
Key elements of the sequence are as follows:

1. The current bus master drives /START to asserted (low), places the desired video RAM space *address* on the /AD31–/AD2 bus, and drives </TM1–/TM0, /AD1–/AD0> with the transfer mode. The /TM1 signal is low when the first transaction in Figure 11-2 starts, indicating that a write transaction is under way.
2. Write output enable (/WROE) is asserted (low).
3. The video card decodes /ID3–/ID0 to determine whether it is in the slot currently being accessed by the current bus master; if so, then the card's /SLOTSEL is asserted.
4. On the next rising (sampling) edge of the video clock (20M), RAM select (/RAMSEL) is asserted; this indicates that a RAM access is to be initiated on the next driving edge of the NuBus clock.

◆ *Note:* The video card clock (20M) is twice the frequency of the NuBus clock (/CLK).

5. The RAM timing chain is commenced, driven by a state machine going sequentially through states 3, 2, 0, 1, and repeating; this machine controls a wait for the data from the bus master/processor to become ready, initiates row and column address strobes, and generates the RAM accesses to do the writing.
6. The bus master drives the /AD31–/AD0 lines with the *data* to be written and releases the /TM1–/TM0 lines and the /ACK line.
7. The video card drives the transaction response status onto the /TM1–/TM0 lines and asserts acknowledge (/ACK), notifying the bus master that the write transaction is completed.
8. The bus master releases the /AD31–/AD0 lines and drives the /ACK line to a determinate state.
9. The video card releases the /TM1–/TM0 lines and also releases /ACK, completing the write transaction.

■ **Figure 11-2** Access to video RAM space



† Start of RAM timing chain.

Color look-up table

The color look-up table (CLUT) is a device that converts the pixel data from the frame buffer into the red, green, and blue video signals. It is actually a combination of a color look-up table and three 8-bit digital-to-analog converters (DACs) integrated into one IC. The CLUT supports up to 256 simultaneous colors from a possible 16.8 million colors.

Color QuickDraw initializes the color-table RAM in the CLUT with default color values using the video driver loaded from the declaration ROM. Color QuickDraw also provides utilities (again by way of the video driver) to read and modify the information in the color table.

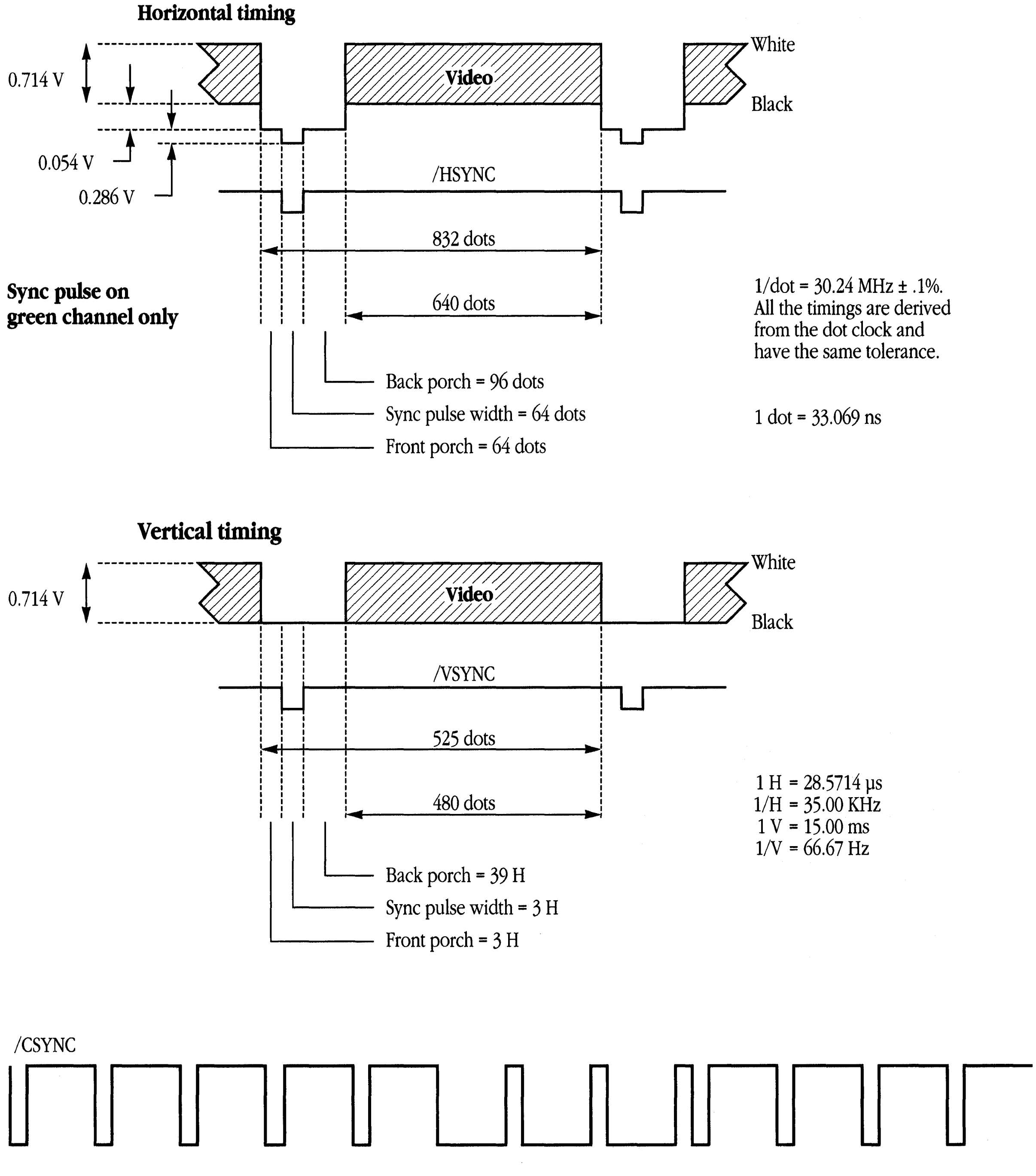
The CLUT is the electrical interface between the FBC and the analog video output device. In operation, the FBC controls the transfer of digital pixel data from the video RAM to the CLUT. Inside the CLUT is a table of RGB triples, one for each currently available color. Each pixel's worth (1, 2, 4, or 8 bits) of data from the frame buffer, acting as an index to this table, selects an RGB triple to be sent through the DACs to the video outputs. The table has storage for 256 RGB triples, enough to support up to 8 bits per pixel.

Each RGB triple from the table consists of three 8-bit values, one each for red, green, and blue. Those values are sent to three 8-bit DACs to generate red, green, and blue analog color signals. The outputs from the DACs provide RS-343-A–compatible or RS-170–compatible RGB video signals to the video connector at the rear of the video card.

Horizontal and vertical scan timing

Figures 11-3 and 11-4 show timing information for the two types of video monitors supported by the video card: Macintosh II–family (high-resolution) RGB and Apple IIGS (RS-170) RGB. These figures define the blanking, synchronizing, and active video regions of the video scan waveforms in terms of dot or pixel times. A dot is the time required to draw a single pixel. H is the time for one horizontal line, including retrace; likewise, V is the time for a vertical scan.

■ **Figure 11-3** Horizontal and vertical scan timing for high-resolution RGB monitor

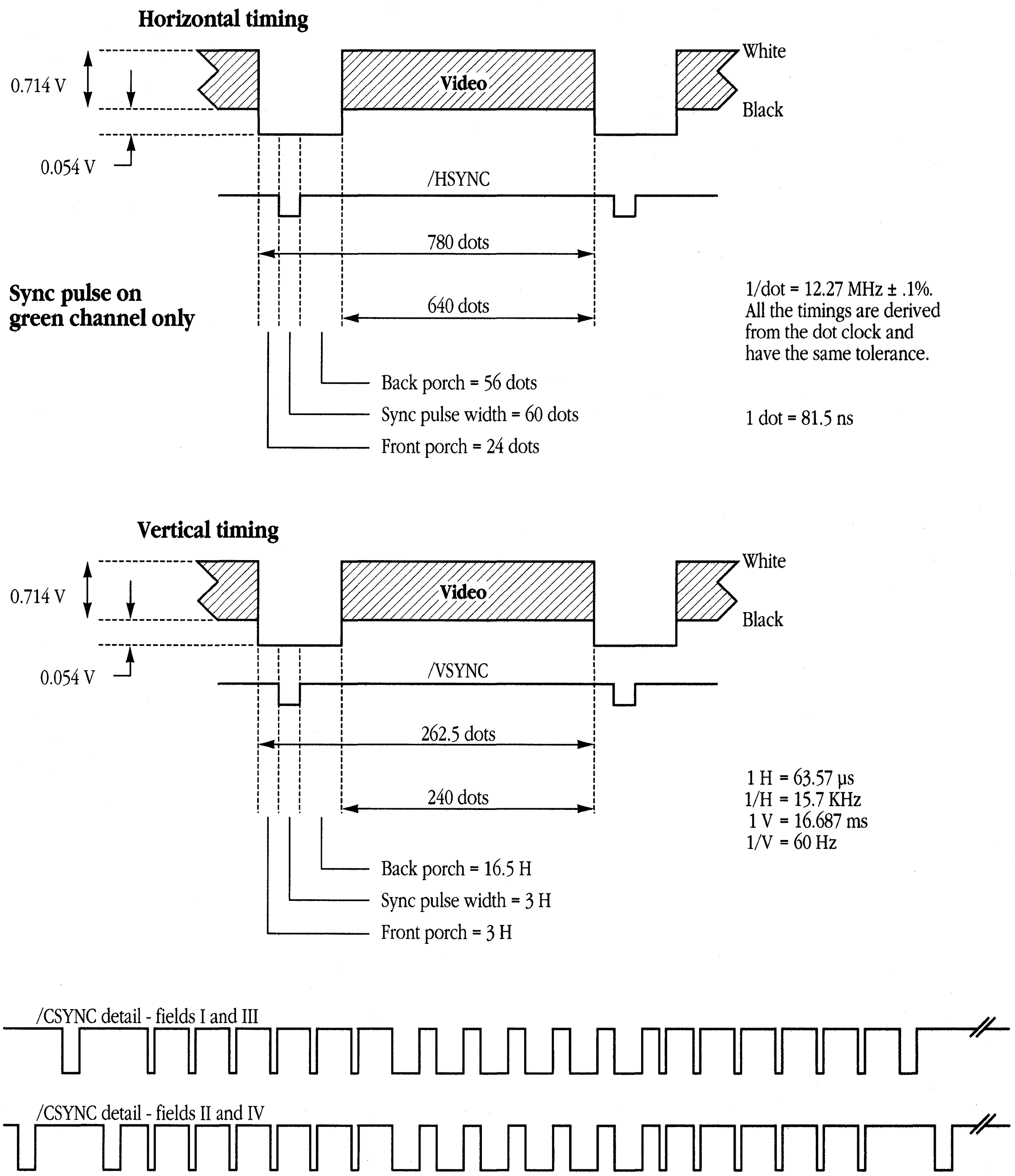


1/dot = 30.24 MHz ± .1%.
All the timings are derived from the dot clock and have the same tolerance.

1 dot = 33.069 ns

1 H = 28.5714 µs
1/H = 35.00 KHz
1 V = 15.00 ms
1/V = 66.67 Hz

■ **Figure 11-4** Horizontal and vertical scan timing for the RS-170 monitor



Declaration ROM operation

The video card includes a declaration ROM that contains all the information the system requires to identify and use the card. The declaration ROM identifies the card as a video device manufactured by Apple and identifies the particular model.

The declaration ROM incorporates three main elements:

- the configuration data
- the drivers
- the primary initialization code

These three elements allow the video card to be installed into a system, recognized, and used without having to run any special configuration programs, and without adding any code to the System file of the host system.

Configuration data

The declaration ROM provides a set of predefined video modes, each element of which specifies all the parameters of the display unique to that mode, including horizontal and vertical size, pixel size, rowbytes of a scan line, and the number of video pages available at this screen resolution.

The video card is highly programmable, and, as a result, the number of possible video modes is enormous. A subset of these video modes, optimized for various Apple display devices, is included in the declaration ROM. The declaration ROM of the Macintosh II Video Card has some unique features. Because the card is available in two configurations—256 KB and 512 KB RAM—a number of mode conflicts arise. Most notably, the 256 KB version of the card does not support 8-bit mode, and each common mode has a different number of video pages available on the two cards. To resolve this problem, the card includes two complete slot resources (sResources), one for the 256 KB card and another for the fully stocked card. For a detailed description of sResources, see Chapter 8, "NuBus Card Firmware."

At startup time, both slot resources are installed in the system's slot resource table. When the primary initialization code of the video card is executed, in addition to initializing the FBC, it performs a size test on the amount of available video RAM and removes the slot resource that does not apply. The 256 KB version of the slot resource list includes configuration information only for 1-, 2-, and 4-bit video modes as well as the appropriate number of video pages available. Normally the video mode of a card in a Macintosh II–family computer is set using a Monitors control panel. See *Inside Macintosh* for more information on Control Panel modules. Monitors finds the available video modes of a video card by examining the declaration ROM's information. By implementing the declaration ROM in the manner described here, a single declaration ROM serves both configurations of the video card without Monitors having to verify device-dependent information (such as memory size).

The driver

The parameter defining each video mode also specifies a software driver, specific to the card hardware and located in the ROM, that is loaded into main memory by the Slot Manager at startup time. This driver is equivalent to the firmware on traditional peripheral cards. Chapter 9, "NuBus Card Driver Design," contains a code listing for a possible video card driver.

◆ *Note:* Because the ROM may not appear on all 4 byte lanes, the driver is loaded into the main memory for execution; object code is not normally executed over the bus.

The Macintosh II High-Resolution Video Card also contains a separate driver specifically designed for video support under the A/UX operating system.

The primary initialization code

The declaration ROM includes a special code, called the **primary initialization code,** that performs key, one-time initialization to the card when executed.

The monitor connected to the high-resolution video card identifies itself by asserting a predetermined combination of signals on the sense lines (SENSE0 and SENSE1 on the video connector). The primary initialization code, executed at system startup, reads the monitor sense lines and selects the appropriate pixel clock rate. Next, after sizing the amount of installed video RAM, the code selects the appropriate sResource and installs it in the Slot Manager's slot device table. This information informs Color QuickDraw about the size and shape of the display, as well as the various pixel depths and number of video pages available on this configuration of card and monitor.

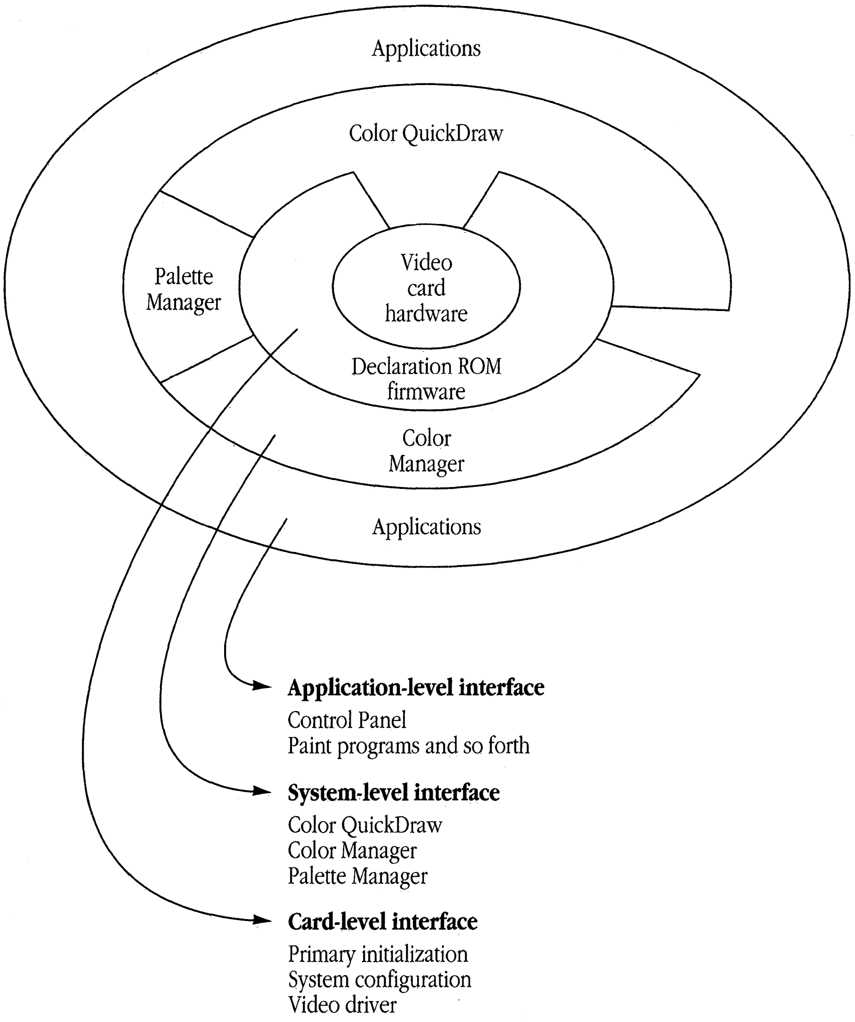
By making this determination at startup time, the primary initialization code permits the system code to be greatly simplified because only information pertinent to the connected monitor is reported by the Slot Manager, and no information about the other type of display is present in the slot device table. This feature greatly simplifies the use of the video card because the display is always correct for the connected monitor, and the monitor cannot be switched into modes where the screen is not readable.

Firmware interfaces

Usually, it is not necessary to access the slot information or the driver directly from the application; instead, Color QuickDraw and the Color Manager in the Macintosh ROM manage all transactions to the card and its driver. Figure 11-5 shows the way those ROM routines mediate between the application and the hardware. For example, the `InitGDevice` routine in Color QuickDraw (documented in the graphics devices information in *Inside Macintosh)* issues all the appropriate calls to change the video mode, load the CLUT, and perform other hardware maintenance tasks, as well as updating system variables pertinent to the affected video device.

Selection of a video mode by the user is made possible by system software such as Monitors and a Control Panel module that graphically presents all possible modes for a video device (as enumerated in the declaration ROM) and allows interactive selection. By always using system code such as QuickDraw and the Control Panel, you will find that applications are simpler to write and present a more uniform interface to the user.

■ **Figure 11-5** Firmware levels



Card connectors

The latest version of the Macintosh II Video Card contains three connectors, one for NuBus, one for video output, and one for external video signals. The connection to the NuBus is through the 96-pin Euro-DIN connector described in Chapters 5 and 6 of this book.

Video connector

The small DB-15 connector at the rear of the card is the video output connector. In addition to the red, blue, and green video output signals and the sync signals, this connector provides the sense lines that enable the card to determine the type of monitor it is connected to. The pinout of the video output connector is shown in Table 11-1.

■ **Table 11-1** Pin assignments for the video output connector

Pin	Signal	Definition
1	GND	Red ground
2	RED	Red video
3	/CSYNCH	Color synchronization
4	SENSE0	Monitor sense line
5	GREEN	Green video
6	GND	Green ground
7	SENSE1	Monitor sense line
8	n.c.	Not connected
9	BLUE	Blue video
10	SENSE2	Monitor sense line
11	GND	Ground
12	/VSYNC	Vertical sync signal
13	GND	Blue ground
14	GND	Ground
15	/HSYNC	Horizontal sync signal

External-signal connector

The external-signal connector is a 14-pin connector that enables the card to accept external sync signals from genlock and overlay cards. Table 11-2 shows the pinout of this connector.

■ **Table 11-2** Pin assignments for the external-signal connector

Pin	Signal	Definition
1	GND	Ground
2	GND	Ground
3	GND	Ground
4	EXTCLK	External clock
5	GND	Ground
6	CLOCKSELECT†	Positive clock select
7	GND	Ground
8	/CBLANK	Blanking
9	GND	Ground
10	/VSYNC	Vertical synchronization
11	GND	Ground
12	/HSYNC	Horizontal synchronization
13	VCC	Power supply voltage
14	/CLOCKSELECT†	Negative clock select

† Note that both polarities of the clock select signal are present.

Part II The Processor-Direct Slot Expansion Interface

About Part II

The processor-direct slot (PDS) expansion interface is the subject of Part II of this book. The seven chapters give you the information you need to design expansion cards for Macintosh PDS computers. Computers that offer PDS expansion capabilities are the Macintosh SE, Macintosh Portable, Macintosh SE/30, Macintosh LC, Macintosh IIfx, Macintosh IIsi, Macintosh Quadra 700, and Macintosh Quadra 900.

Chapter 12 compares the major features of the Macintosh PDS computers and gives a general overview of their operation. The chapter provides block diagrams of the Macintosh SE, Macintosh Portable, Macintosh SE/30, and Macintosh LC computers, and then describes the capabilities of the PDS expansion interface.

Chapter 13 contains the electrical information you need to design cards for the 68000 Direct Slot. The Macintosh SE and the Macintosh Portable both offer the 68000 PDS expansion capabilities. Chapter 13 provides electrical design guidelines for the 68000 Direct Slot, including connector pinouts and signal descriptions, expansion card load limits and drive requirements, instructions for accessing the computer electronics, address spaces, and power consumption guidelines.

Chapter 14 discusses the electrical design guidelines you need to design cards for the 68020 Direct Slot. The Macintosh LC offers a 68020 PDS expansion slot. The topics in Chapter 14 include a discussion of the 68020 Direct Slot expansion connector pinouts and signal descriptions, expansion connector load limits and drive requirements, power consumption guidelines, memory and I/O access from an expansion card, a discussion of pseudoslot design, and specific hints to be used for your 68020 expansion card.

Chapter 15 provides electrical guidelines for designing cards for the 68030 Direct Slot for the Macintosh SE/30, the Macintosh IIfx, and the Macintosh IIsi. Topics include the 68030 Direct Slot expansion connector pinouts and signal descriptions, expansion connector load limits and drive requirements, information on accessing the computer's main logic board, I/O, and memory devices from an expansion card, pseudoslot design information, specific hints for developers, and power consumption guidelines.

Chapter 16 includes electrical design guidelines for the 68040 Direct Slot in the Macintosh Quadra 700 and Macintosh Quadra 900 computers. Topics covered in Chapter 16 include an electrical description of the 68040 expansion connector, the connector pinouts and the signal descriptions, access to memory and I/O devices from 68040 Direct Slot expansion cards, pseudoslot information, cache management techniques, interrupt handling for the expansion cards, power consumption guidelines, and design hints for the 68040 Direct Slot expansion cards.

Chapter 17 details the physical information you need to design Macintosh PDS expansion cards for each of the current Macintosh PDS computers.

Chapter 18 describes a proven design of a simple disk controller card that uses the Macintosh SE 68000 Direct Slot.

Chapter 12 **Overview of Macintosh PDS Computers**

This chapter provides an overview of the structure and organization of the Macintosh family of computers that use a single processor-direct slot (PDS) as their primary expansion interface. Included in this category are the Macintosh SE, the Macintosh Portable, the Macintosh SE/30, and the Macintosh LC. The PDS expansion interface relates directly to the microprocessor it supports. The Macintosh SE and the Macintosh Portable are configured with 96-pin 68000 Direct Slots, the Macintosh LC with a 96-pin 68020 Direct Slot, and the Macintosh SE/30 with a 120-pin 68030 Direct Slot.

Certain Macintosh computers use the NuBus as their primary expansion interface and are not discussed here even though they offer PDS expansion slots. The hardware overview of these computers—the Macintosh IIfx, the Macintosh IIsi, the Macintosh Quadra 700, and the Macintosh Quadra 900—is provided in Chapter 1, "Overview of Macintosh Computers With the NuBus Interface."

This chapter places the internal microprocessor expansion bus and PDS connector in context within the total computing machine. Subsequent chapters provide the information needed to design expansion cards compatible with the PDS configuration. This chapter assumes you're familiar with the basic operation of microprocessor-based devices.

Major features

Table 12-1 compares the major features of all Macintosh computers that use the processor-direct slot as their primary expansion interface.

■ **Table 12-1** Major features of Macintosh computers with processor-direct slots

Feature	Macintosh SE	Macintosh Portable	Macintosh SE/30	Macintosh LC
Processor	MC68000 24-bit address bus, 16-bit data bus	MC68HC000 24-bit address bus, 16-bit data bus	MC68030 32-bit address bus, 32-bit data bus	MC68020 29-bit address bus, 16-bit RAM data bus, 32-bit ROM data bus
Auxiliary processor	Not applicable	Power Manager IC keyboard processor	Not applicable	Not applicable
Processor clock	7.8336 MHz	15.6672 MHz	15.6672 MHz	15.6672 MHz
Coprocessor	Not applicable	Not applicable	MC68882 FPU	FPU possible on expansion slot
Memory management	Not applicable	Not applicable	MC68030 includes a built-in PMMU that allows true 32-bit address translation with hardware page replacement	Not applicable
RAM	1 MB, expandable to 4 MB	1 MB, expandable to 5 MB, using SRAM, PSRAM, and DRAM	1 MB or 4 MB, expandable to 8 MB (expandable to 128 MB when higher-density DRAM chips are available)	2 MB, expandable to 4 MB or 10 MB; 256 KB or 512 KB of VRAM
ROM	256 KB	256 KB	256 KB	512 KB, optional expansion to 4 MB
Expansion slot	68000 Direct Slot, 96-pin	68000 Direct Slot, 96-pin	68030 Direct Slot, 120-pin	68020 Direct Slot, 96-pin
Input device interface	Two Apple Desktop Bus (ADB) ports for keyboard, mouse, or optional input device	Built-in alphanumeric keyboard and trackball; ADB for optional input devices	Two ADB ports for keyboard, mouse, or optional input device	One ADB port, one audio-input jack for microphone or line input

(continued)

■ **Table 12-1** Major features of Macintosh computers with processor-direct slots (continued)

Feature	Macintosh SE	Macintosh Portable	Macintosh SE/30	Macintosh LC
Serial ports	Two mini 8-pin connectors supporting RS-422	Two mini 8-pin connectors supporting RS-422	Two mini 8-pin connectors supporting RS-422	Two mini 8-pin connectors supporting RS-422
Floppy disk support	Two internal floppy disk drives, one standard, one optional. Super Woz Integrated Machine (SWIM) controls two 3.5", 1.4 MB SuperDrives; earlier models used Integrated Woz Machine (IWM) to control two 3.5", 1.4 MB SuperDrives	SWIM controls two internal 1.4 MB, 3.5" SuperDrives (one standard, one optional); external floppy disk drive port	SWIM controls internal 1.4 MB, 3.5" SuperDrive; external floppy disk drive port	SWIM controls two internal 3.5", 1.4 MB SuperDrives (one standard, one optional)
Hard disk	Optional internal 20 or 40 MB SCSI hard disk; optional external SCSI hard disk	Optional internal 40 MB SCSI hard disk; optional external SCSI hard disk	Internal 40 or 80 MB SCSI hard disk, optional external SCSI hard disk	Optional internal 40 MB SCSI hard disk, optional external SCSI hard disk
SCSI port	One internal 50-pin; one external DB-25	One internal 34-pin; one external DB-25	One internal 50-pin; one external DB-25	One internal 50-pin; one external DB-25
Sound	Standard Macintosh sound chip	Custom Apple Sound Chip (ASC)	Custom ASC	V8 gate array provides a subset of ASC implementation
Video display	Built-in 9" monochrome monitor, 512 × 342 pixels	Built-in LCD, 9.8" flat panel, 640 × 400 pixels	Built-in 9" monochrome monitor, 512 × 342 pixels	Optional 12" monochrome display, 12" RGB display, or 13" AppleColor High Resolution display
Battery	Long-life lithium battery backup	Rechargeable, 8-hour, lead-acid battery retains RAM contents during sleep state; 9 V battery holds RAM contents while the primary battery is being replaced	Long-life lithium battery backup	Long-life lithium battery backup

Hardware architecture

The following discussion is brief and intended primarily to show the place of the processor-direct slot (PDS) expansion connector in the machine architecture. For a complete description of hardware operation, see the *Guide to the Macintosh Family Hardware.* Also useful would be the *Macintosh IIsi, LC, and Classic Developer Notes* and the *Macintosh Classic II, Macintosh PowerBook Family, and Macintosh Quadra Family Developer Notes.* Or, if you are interested in a higher-level overview, see the *Technical Introduction to the Macintosh Family.*

The Macintosh SE and the Macintosh SE/30 are similar in appearance to the original Macintosh computer. The Macintosh SE contains a Motorola MC68000 microprocessor operating at 7.8336 MHz and a Euro-DIN 96-pin connector for hardware expansion. The Macintosh Portable also has a Euro-DIN 96-pin expansion connector, but it is electrically different from the expansion connector used on the Macintosh SE. In addition, the Macintosh Portable has special low-power components throughout, including an MC68HC000 microprocessor operating at 15.6672 MHz, and it incorporates a built-in flat panel liquid crystal display (LCD).

The architecture of the Macintosh LC is based on the Macintosh IIci architecture and uses a Motorola MC68020 microprocessor operating at 15.6672 MHz and a Euro-DIN 96-pin connector for hardware expansion. The expansion connector, however, is electrically different from that found in the Macintosh SE and the Macintosh Portable.

The Macintosh SE/30 is similar in external appearance to the Macintosh SE; and although the interior is also very similar in appearance, the components on the main circuit board of the Macintosh SE/30 are more closely related to those of a Macintosh IIx. The Macintosh SE/30 has a Motorola MC68030 microprocessor that operates at 15.6672 MHz and a Euro-DIN 120-pin connector for hardware expansion. Block diagrams of the Macintosh SE, Macintosh Portable, Macintosh SE/30, and Macintosh LC computers are shown in Figures 12-1 through 12-4.

These PDS computers contain several common circuits, including random-access memory (RAM), read-only memory (ROM), and some I/O chips that enable the microprocessor to communicate with external devices. Following is a brief description of these I/O chips:

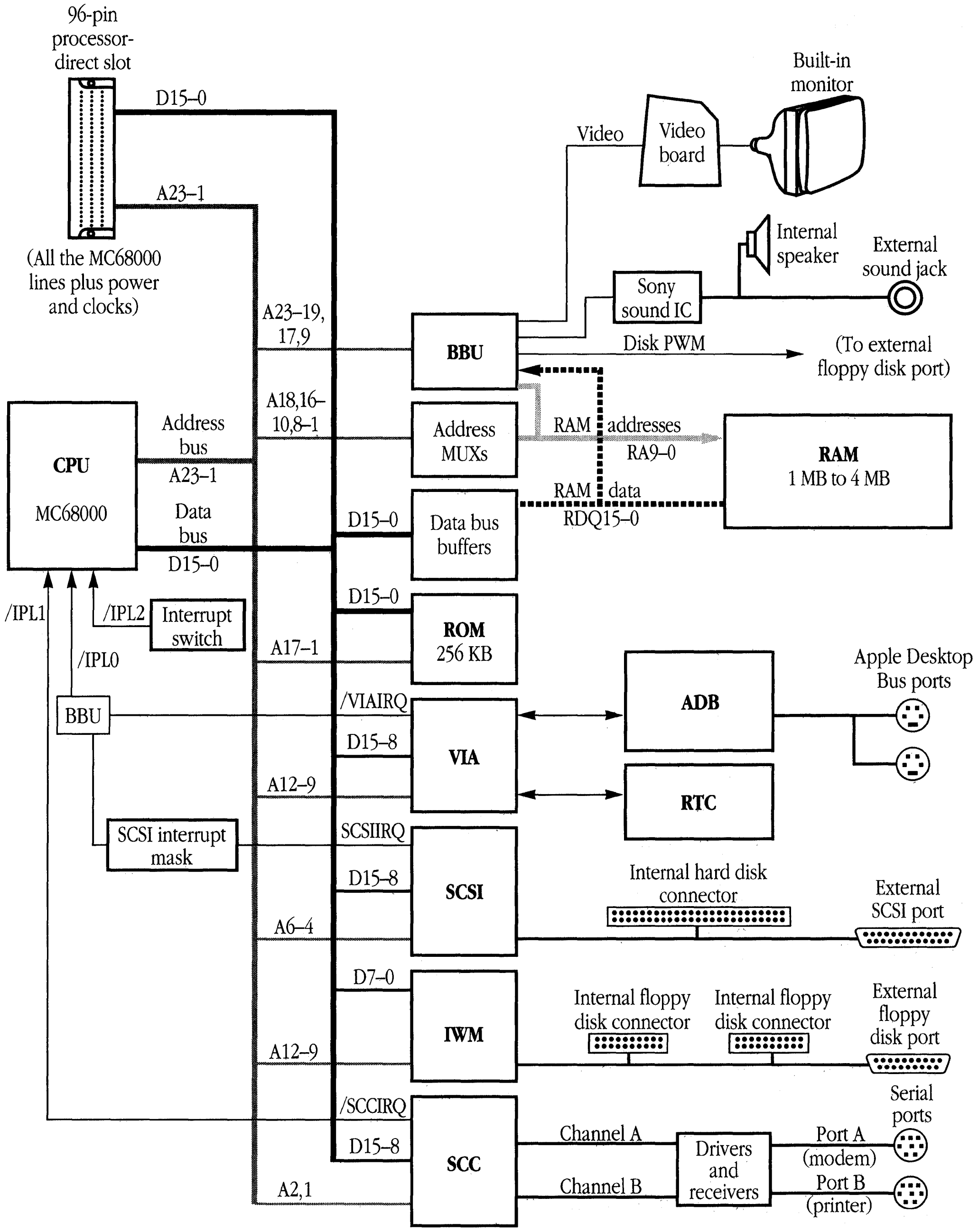
- Every Macintosh computer except the Macintosh LC has one or two Apple custom Versatile Interface Adapter (VIA) chips. The Macintosh SE and the Macintosh Portable each have one VIA chip. The VIA in the Macintosh SE supports the Apple Desktop Bus (ADB) and the real-time clock (RTC). The VIA in the Macintosh Portable provides the communication interface between the processor and the Power Manager IC as well as interrupts for a number of internal functions. The Macintosh SE/30 has two VIA chips, VIA1 and VIA2. VIA1 supports the same functions as the Macintosh SE VIA, while VIA2 supports features such as expansion card interrupts, Apple Sound Chip interrupts, and others.
- The Macintosh LC uses a new custom VLSI chip, the V8 gate array, to integrate timing, address decode, video generation, clock generation, sound control, and GLU (general logic unit) functions that were provided by individual chips in other Macintosh computers.
- A SCSI (Small Computer System Interface) chip provides high-speed parallel communication with internal or external devices such as hard disks.
- A Serial Communications Controller (SCC) provides for high-speed, asynchronous serial communication (also synchronous modem support). In the Macintosh LC, a custom chip, Combo, combines the functions of the SCC and the SCSI controller in a single device. This device is completely software compatible with the SCC and SCSI chips it replaces.
- An Apple custom chip controls both internal and external floppy disk drives. Earlier Macintosh SE models used an IWM (Integrated Woz Machine) chip to control 3.5-inch, 800 KB floppy disk drives. More recent models of the Macintosh SE, as well as the Macintosh SE/30, the Macintosh LC, and the Macintosh Portable, use the SWIM (Super Woz Integrated Machine) chip to control 3.5-inch, 1.4 MB high-density disk drives.

- The Macintosh SE includes an Apple custom chip, called the BBU (Bob Bailey Unit), for video and sound control and for generating device-select signals. The Macintosh SE/30 and the Macintosh Portable use the custom Apple Sound Chip (ASC) to control stereo sound and other enhancements not available on the Macintosh SE.
- The Macintosh Portable includes a custom integrated circuit called the Power Manager IC that controls the distribution of power to all I/O devices. Not all devices can be addressed directly, and those that can require Power Manager IC cooperation to ensure that power will be applied during the access time.
- The Macintosh LC also uses a microcontroller that integrates the functions of ADB interface, real-time clock, power-on reset, parameter RAM storage, keyboard-controlled reset, and NMI (nonmaskable interrupt). This microcontroller is also used in the Macintosh IIsi. On other Macintosh computers, the reset and NMI functions are hardware controlled by the programmer's switch and the reset switch. The ADB microcontroller also controls the DFAC (Digitally Filtered Audio Chip), a custom IC that performs the analog processing functions of the sound system.

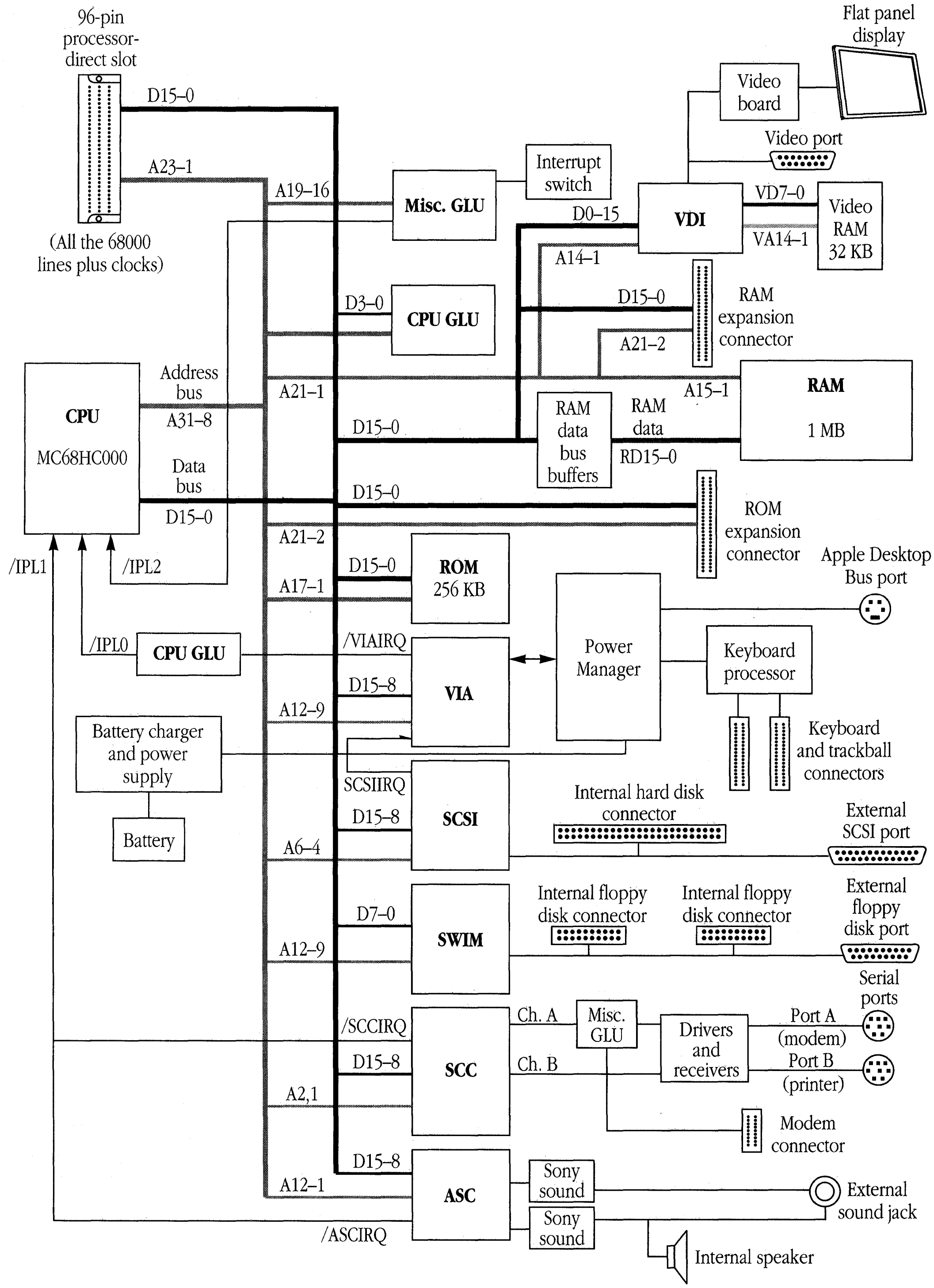
All Macintosh computers use memory-mapped I/O, which means that you can gain access to each device in the system by reading from or writing to specific locations in the address space of the computer.

The MC68000 and the MC68HC000 processors used in the Macintosh SE and the Macintosh Portable can directly access 16 M of address space. The MC68020 processor used in the Macintosh LC and the MC68030 processor used in the Macintosh SE/30 can directly access 128 MB of address space. This address space is divided into several areas allocated to RAM, ROM, and various I/O devices.

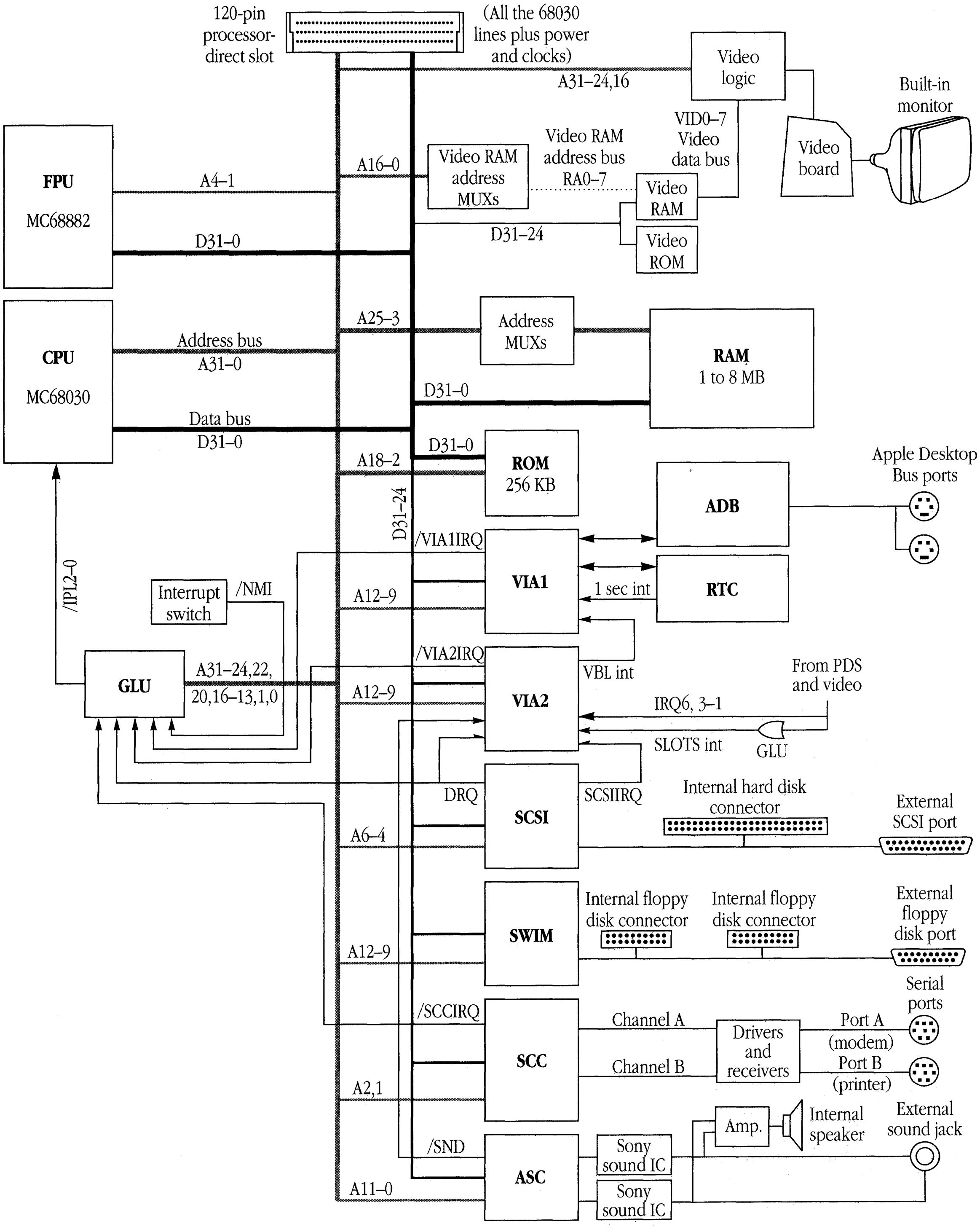
■ **Figure 12-1** Block diagram of the Macintosh SE computer



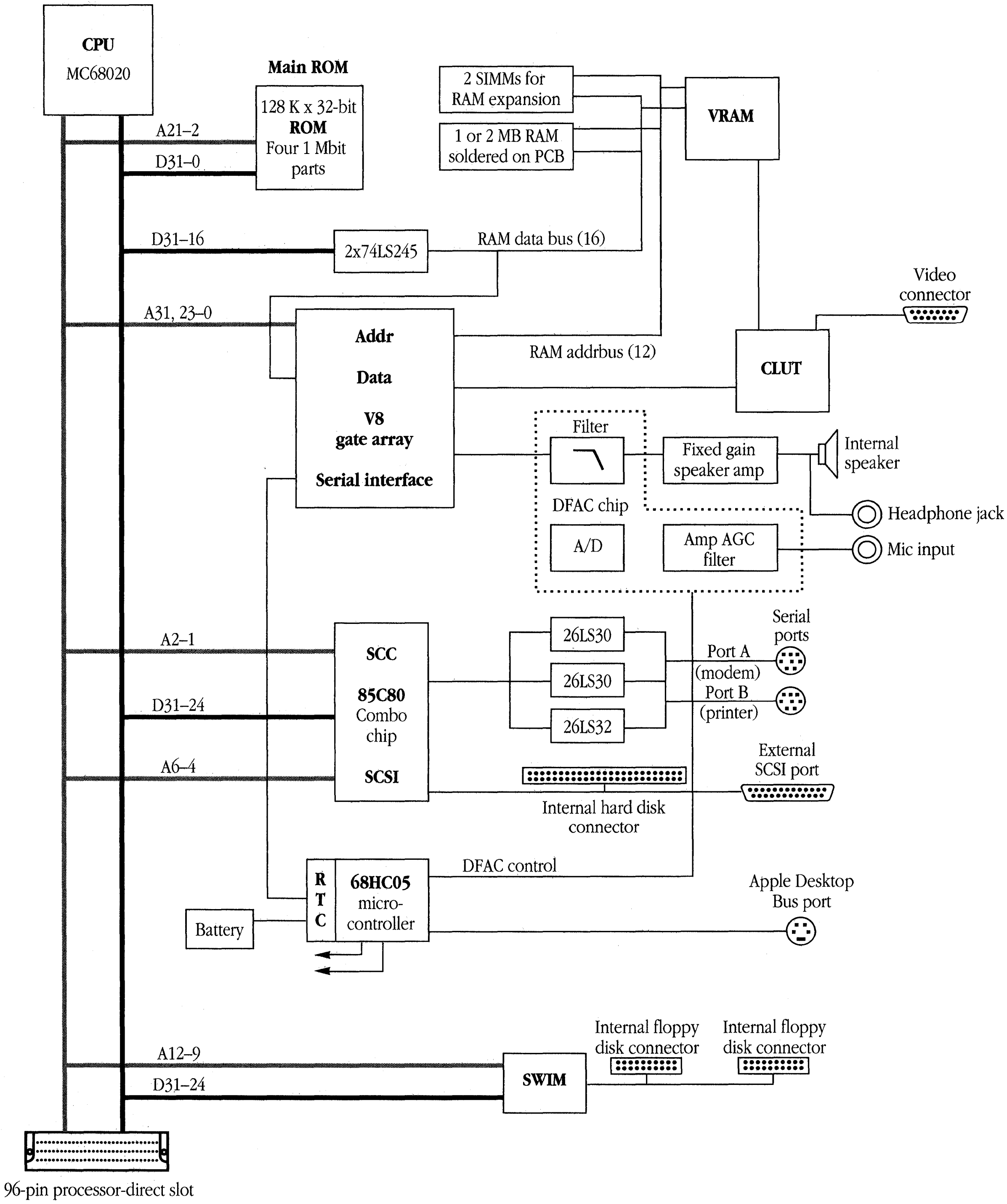
■ **Figure 12-2** Block diagram of the Macintosh Portable computer



■ **Figure 12-3** Block diagram of the Macintosh SE/30 computer



■ **Figure 12-4** Block diagram of the Macintosh LC computer



RAM

RAM is the working memory of the system. In the Macintosh SE computer, address space from $00 0000 through $3F FFFF is reserved for RAM. In the Macintosh Portable computer, address space from $00 0000 through $8F FFFF is reserved for RAM. Address space $0000 0000 through $3FFF FFFF is reserved for RAM in the MC68030-based Macintosh computers. In the Macintosh LC, address space from $00 0000 through $9F FFFF is reserved for RAM. The actual amount of address space used depends upon the amount of RAM available in the system.

The processors in the Macintosh computers use the first 1024 bytes of RAM (addresses $00 0000 through $00 03FF) as exception vectors; these are the addresses of the routines that gain control whenever an exception such as an interrupt or a trap occurs. The vector base register (VBR) also points to the beginning of the exception vector table. The first 256 bytes of the exception vectors are reserved for use by the operating system, and the remainder are allocated for use by applications. RAM also contains the system and application heaps, the stack, a copy of parameter RAM, various global variables and trap handlers, and other information used by applications.

In addition, the following hardware devices share the use of RAM with the MC68000 on the Macintosh SE:

- the video display, which reads the information for the display from one of two screen buffers
- the sound generator, which reads its information from a sound buffer
- the disk-speed controller (used only with an external, single-sided floppy disk drive), which shares its data space with the sound buffer

The Macintosh Portable has separate RAM buffers for sound and video and does not have a disk-speed controller; therefore, it does not share its system RAM with other devices.

In the Macintosh SE, the processor's accesses to RAM are interleaved with the video display's accesses. In the Macintosh SE/30 and Macintosh Portable, the processor's accesses to RAM are not interleaved with the video display's accesses because the video circuitry includes memory that is used exclusively by the video display.

The Macintosh LC is shipped with 256 KB or 512 KB of VRAM (video RAM). With this configuration, main memory is not used for storing video data. If VRAM is not installed, it is possible to use main memory for video storage, though it is not recommended. With this configuration, main memory can only support the 640 × 480 monochrome video mode.

ROM

ROM is the system's permanent read-only memory. When the Macintosh is first turned on, a second image of ROM appears at $00 0000, so that ROM can supply the processor with the exception vectors. Following the first access to the normal address ranges of ROM or the SCSI controller, the image of ROM at $00 0000 is replaced by RAM.

The base address of ROM is stored in the global variable `ROMBase`. ROM contains the routines for the User Interface Toolbox and the Macintosh Operating System, and the various system traps.

Device I/O

Macintosh computers use memory-mapped I/O, which means that each device in the system is accessed by reading from or writing to specific locations in the address space of the computer. The address space reserved for the device I/O contains blocks devoted to each of the devices within the computer. Each device contains logic that recognizes when it's being accessed, and the device responds in the appropriate manner. Refer to the section "Device I/O" in Chapter 1, which covers this topic in more detail.

PDS expansion interface

The PDS expansion interface has been designed to help hardware developers add reliable and elegant custom hardware to the Macintosh family of computers.

Following the design guidelines in Chapters 13 through 17, you may choose to offer cards such as the following:

- custom video card
- network communication interface card
- modem card
- coprocessor or accelerator card

The foregoing list is not intended to limit or authorize, in any way, the types of expansion cards that you may want to develop.

The 68000 Direct Slot

The Macintosh SE was the first Macintosh computer to offer PDS expansion. A Euro-DIN 96-pin connector on the main circuit board provides unbuffered access to the MC68000 processor bus, including all address, data, and control lines. In addition, extensive power and grounding are provided in the expansion connector, as well as critical high-speed timing signals. The 68000 Direct Slot supports high-speed direct memory access into the RAM, allows coprocessors to share the address and data bus, and allocates generous portions of the address space for new peripherals. An expansion card in the 68000 Direct Slot can access system RAM and ROM at the same rates as the MC68000 microprocessor. RAM accesses occur at 3.22 MB per second, and ROM accesses are at 3.92 MB per second.

The physical design of a Macintosh SE permits you to mount an expansion card of approximately 4 inches by 8 inches in area in a position horizontal to the main board. The 96-pin expansion connector provides one mounting point for the expansion card, and there are holes at the opposite side of the main logic board for two mounting posts. Both the Macintosh SE logic board and the chassis have been designed to allow mounting and removal of the logic board while it is joined to an expansion card.

Chapter 17 describes the physical provisions for mounting an expansion card in a Macintosh SE 68000 Direct Slot. See Figures 17-1 through 17-8 for drawings of these mounting provisions.

The Macintosh Portable has the same Euro-DIN 96-pin connector as the Macintosh SE, but connector pinouts are different, and there is no provision for accessing internal hardware signals from outside the Macintosh Portable case. The expansion interface connector on the Macintosh Portable is also referred to as a 68000 Direct Slot even though this machine uses the MC68HC000 processor—a high-speed, low-power, CMOS version of the MC68000 processor. Although this connector is available for expansion purposes, there are certain limitations that may restrict you in designing expansion cards. These limitations are described in Chapter 13 in the section "68000 Direct Slot Expansion for the Macintosh Portable."

The 68020 Direct Slot

A Euro-DIN 96-pin socket connector provides the PDS expansion for the Macintosh LC. This connector is physically, but not electrically, identical to the PDS connector used on the Macintosh SE. The processor-direct expansion connector on the Macintosh LC provides access to the MC68020 microprocessor's full 32-bit data bus and 29 address lines, as well as to a selection of control signals.

The expansion card mounts parallel to the main logic board, component side facing component side. The features that you implement in your design are limited only by the size of the card, approximately 3 inches by 5.4 inches, and the available power. The 96-pin expansion connector provides one mounting point for the expansion card, and there are holes at the opposite side of the main logic board for two mounting posts.

An opening in the rear of the Macintosh LC case allows an expansion card to communicate with external devices. This opening accommodates a DB-15 connector, which you can include as an integral part of your expansion card design.

Chapter 17 describes the physical design guidelines for mounting an expansion card in the Macintosh LC 68020 Direct Slot.

The 68030 Direct Slot

The 68030 Direct Slot expansion connector supports the 32-bit address and data bus features of the MC68030 microprocessor. The expansion hardware consists of a 120-pin Euro-DIN expansion connector that provides access to the MC68030 processor's address and data bus signals, DMA and other processor control signals, interrupt signals, status signals, and power and grounding for the expansion card.

The 68030 Direct Slot expansion feature is provided on the Macintosh SE/30, the Macintosh IIfx, and the Macintosh IIsi. Though all three Macintosh computers use the 120-pin expansion connector, there are some differences. The Macintosh IIsi requires that a 68030 Direct Slot adapter card be installed first before installing a processor-direct expansion card. The expansion connector on the 68030 Direct Slot adapter card is physically and electrically identical to the expansion connector found on the Macintosh SE/30 main logic board. This allows expansion cards to work in both the Macintosh IIsi and the Macintosh SE/30 without change, as long as the expansion cards can run at the different clock speeds of both computers.

The Macintosh IIfx also uses the 120-pin expansion connector; however, it differs electrically from the Macintosh IIsi and the Macintosh SE/30. Chapter 15 discusses the electrical characteristics of the 68030 Direct Slot expansion. Chapter 17 provides information about the physical characteristics of installing a PDS expansion card. Also covered in these two chapters is a discussion of the Macintosh IIsi adapter card.

The 68040 Direct Slot

On the Macintosh Quadra 700 and the Macintosh Quadra 900 computers, a 140-pin PDS expansion connector is used to provide unbuffered access to the MC68040 microprocessor's signals by way of the system bus. The system bus on the Macintosh Quadra 700 and the Macintosh Quadra 900 is a high-performance synchronous bus that runs at the same clock speed as the 68040 bus clock and includes new features such as burst read and write. The PDS expansion connector supports the 32-bit address and data bus features of the MC68040 microprocessor.

The 68040 PDS expansion slots for the Macintosh Quadra 700 and the Macintosh Quadra 900 are pin-for-pin compatible. Any expansion cards developed for the Macintosh Quadra 700 also work in the Macintosh Quadra 900.

In the Macintosh Quadra family of computers, a PDS card has the same dimensions as a NuBus card. Because of this, a PDS card can be larger for the Macintosh Quadra 900. The oversized card for the Macintosh Quadra 900 is approximately 6 inches high and between 7 and 12 inches wide. The size of the Macintosh Quadra 700 PDS expansion card, however, must be smaller, the same size as a standard NuBus card. Foldouts 1, 2, and 5, included at the back of this book, describe the dimensions for standard and oversized NuBus cards.

The PDS expansion cards for both the Macintosh Quadra 700 and the Macintosh Quadra 900 can include a back-panel connector. For more information about physical design guidelines for the 68040 PDS expansion cards, please refer to Chapter 17.

Additional support for expansion

The Macintosh SE and the Macintosh SE/30 have power supplies and fans that are designed to provide additional power and cooling for the electronics on expansion cards.

Both the Macintosh SE and the Macintosh SE/30 include a feature that allows cables to be routed from an expansion card to a bracket and an access opening at the rear of the case. The bracket can hold custom connectors on a small connector board that may also contain filter electronics. Chapter 17 contains drawings showing how to connect an expansion card to external devices through the external device access opening.

Third-party products that adhere to the expansion guidelines in Chapters 13 through 17 and Appendix A, that use the Apple-supplied expansion features, and that do not require physical alteration of the computer will not void the Apple Limited Warranty.

Motorola has extensively documented its MC68000 family of microprocessors. For a more detailed understanding of the interface between your expansion card and the microprocessor bus, please refer to the following documents:

- *MC68000 16/32-Bit Microprocessor User's Manual,* Motorola document AD1814R5, March 1985
- *MC68020 32-Bit Microprocessor User's Manual,* Motorola document MC68020UM/AD
- *MC68030 Enhanced 32-Bit Microprocessor User's Manual,* Motorola document MC68030UM/AD
- *MC68040 32-Bit Microprocessor User's Manual,* Motorola document MC68040UM/AD

In summary, PDS expansion is supported by these features:

- Euro-DIN type expansion connector (96-pin on the Macintosh SE, Macintosh Portable, and Macintosh LC; 120-pin on the Macintosh SE/30, Macintosh IIsi, and Macintosh IIfx; 140-pin on the Macintosh Quadra 700 and Macintosh Quadra 900) that provides power, timing, and direct access to the computer's microprocessor bus
- stand-off mounting for card physical support
- main logic board layout and installation features improved from earlier Macintosh models
- external device access opening (Macintosh SE, Macintosh SE/30, Macintosh LC, Macintosh Quadra 700, and Macintosh Quadra 900) provided at rear of case for installation of custom external connector

Chapter 13 **Electrical Design Guide for 68000 Direct Slot Expansion Cards**

This chapter provides the electrical information you need to design expansion cards for Macintosh computers with the 68000 Direct Slot expansion interface. The chapter covers the following topics:

- electrical description of the 68000 Direct Slot expansion connectors for the Macintosh SE and the Macintosh Portable
- signal mnemonics and descriptions
- accessing the Macintosh SE electronics from an expansion card
- available address space
- power consumption guidelines

68000 Direct Slot expansion for the Macintosh SE

This section gives the pinouts and describes the signal characteristics of the 68000 Direct Slot expansion connector used on the Macintosh SE. Information and timing diagrams show you how to access the computer's electronics from the expansion card. This section also discusses available address space and describes the additional power required to operate an expansion card in a Macintosh SE computer. Physical guidelines for designing a Macintosh SE PDS expansion card are provided in Chapter 17.

Electrical description of the Macintosh SE expansion connector

Figure 13-1 gives the pinout for the 96-pin expansion connector (socket) on the Macintosh SE main logic board, as viewed from above.

Table 13-1 gives signal descriptions and the load presented, or drive available, to each pin on an expansion card inserted into the 96-pin expansion connector.

The last column in Table 13-1, labeled "Loading or Driving Limits," gives several specifications. An example may be helpful in interpreting this column. The /RESET line is shown as presenting a load of 300 μA/6 mA, 50 pF. This is the maximum expected load that an expansion card must drive when sending a /RESET signal to the main logic board. The DC load is given in the format *signal high/signal low.* This means that the expansion card driver must drive a load of up to 300 μA when it drives /RESET high (logic 1), and a load of up to 6 mA when it drives /RESET low (logic 0). The AC load is given as 50 pF, the maximum capacitance to ground presented by the main logic board to AC signals (or signal transitions) from the expansion card. The notation "Open collector; 1 kΩ pull-up" in the table means that the /RESET line is normally in the open collector state: it is *only* driven low, and a 1 kΩ pull-up resistor on the main logic board returns the line to a logic 1.

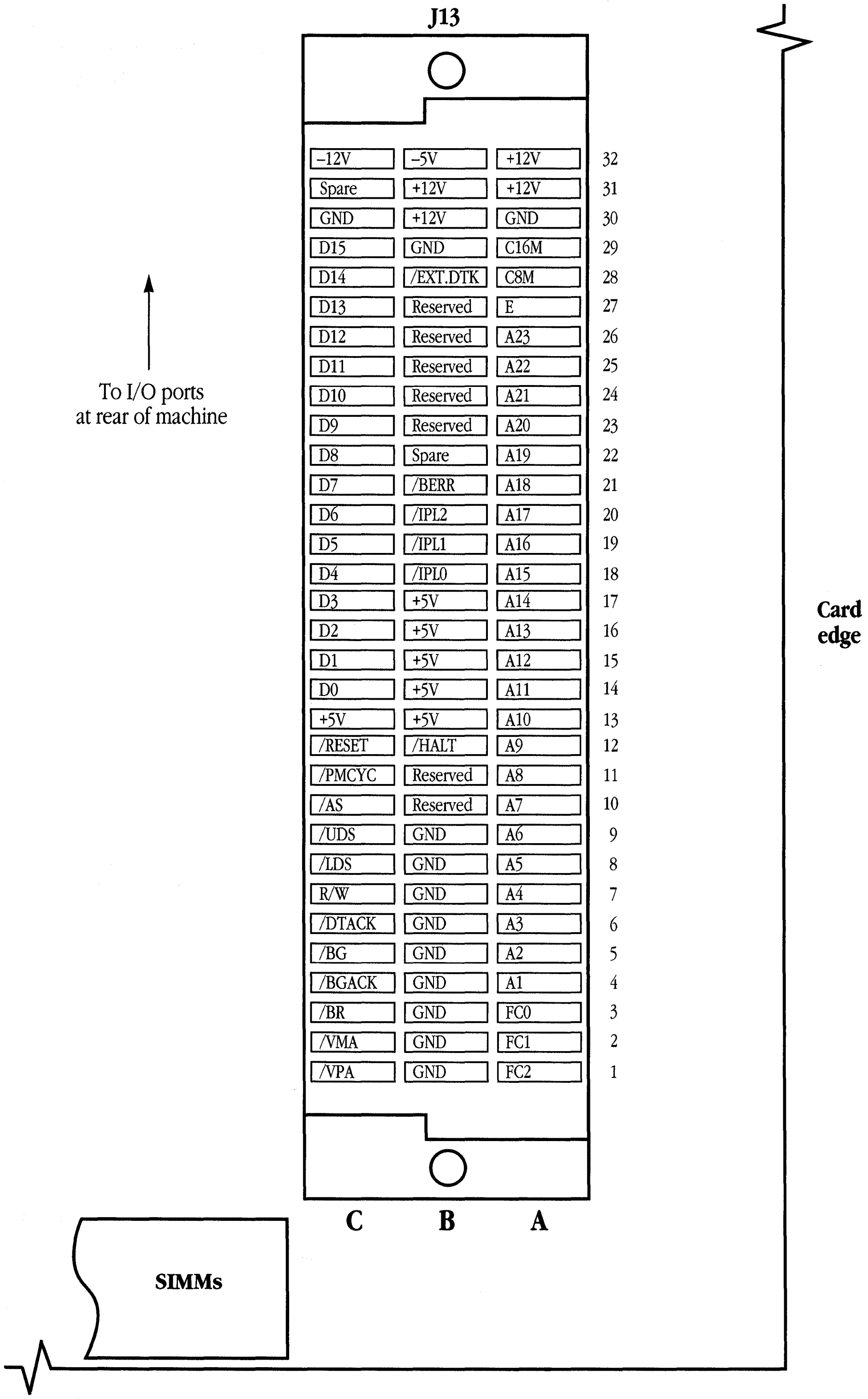
Correspondingly, /RESET presents a drive of 40 µA/0.4 mA, 30 pF. This is the maximum amount of drive from the main logic board that is available to receiving integrated circuits on an expansion card. The /RESET line can drive an expansion card DC load of up to 40 µA in the high (logic 1) state, or up to 0.4 mA in the low (logic 0) state. The AC drive is given as 30 pF, the maximum capacitance to ground that an expansion card may present to AC signals (or signal transitions) from the /RESET line.

The C8M and C16M clock outputs are specified to drive one 74LS input (a standard 74LS input load is 20 µA high, 0.2 mA low) and 20 pF. All other outputs have been specified to drive two 74LS inputs, and 30 pF.

In most cases, these drive limitations are imposed to protect the noise and timing margins of the main logic board. Expansion cards requiring more drive, or more than about 2 inches of trace length, should buffer these signals before distributing them to the effective loads on the card or to external devices connected through the external device access opening.

Where "Load:" is in parentheses, the pin carries a signal that is usually an output driven by the MC68000 but that is tristated by the MC68000 after responding to a bus request. When tristated by the MC68000, this pin may be driven by an expansion card.

■ **Figure 13-1** Macintosh SE 68000 Direct Slot connector pinout



C	B	A	
−12V	−5V	+12V	32
Spare	+12V	+12V	31
GND	+12V	GND	30
D15	GND	C16M	29
D14	/EXT.DTK	C8M	28
D13	Reserved	E	27
D12	Reserved	A23	26
D11	Reserved	A22	25
D10	Reserved	A21	24
D9	Reserved	A20	23
D8	Spare	A19	22
D7	/BERR	A18	21
D6	/IPL2	A17	20
D5	/IPL1	A16	19
D4	/IPL0	A15	18
D3	+5V	A14	17
D2	+5V	A13	16
D1	+5V	A12	15
D0	+5V	A11	14
+5V	+5V	A10	13
/RESET	/HALT	A9	12
/PMCYC	Reserved	A8	11
/AS	Reserved	A7	10
/UDS	GND	A6	9
/LDS	GND	A5	8
R/W	GND	A4	7
/DTACK	GND	A3	6
/BG	GND	A2	5
/BGACK	GND	A1	4
/BR	GND	FC0	3
/VMA	GND	FC1	2
/VPA	GND	FC2	1

■ **Table 13-1** Macintosh SE 68000 Direct Slot signals, loading or driving limits

Connector		Signal	Signal	Input or	Loading or driving
Row	**Pin**	**name**	**description**	**output**	**limits (high/low)**
A	1	FC2	Function code 2	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
A	2	FC1	Function code 1	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
A	3	FC0	Function code 0	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
A	4–26	A1–23	Address 1–23	In/Out	Load: 250 µA/1 mA, 100 pF Drive: 40 µA/0.4 mA, 30 pF
A	27	E	E (enable) clock	Output	Drive: 40 µA/0.4 mA, 30 pF
A	28	C8M	7.8336 MHz MC68000 clock	Output	Drive: 20 µA/0.2 mA, 20 pF
A	29	C16M	15.6672 MHz gate array and IWM clock	Output	Drive: 20 µA/0.2 mA, 20 pF
A	30	GND	Logic ground		
A	31	+12V	+12 volts	Output	Drive: 150 mA total, from all +12V pins
A	32	+12V	+12 volts	Output	(See the section “Power Consumption Guidelines for Macintosh SE PDS Expansion Cards” later in this chapter.)
B	1–9	GND	Logic ground		
B	10	Reserved	For future Apple use; do not connect		
B	11	Reserved	For future Apple use; do not connect		
B	12	/HALT	MC68000 halt	In/Out	Load: 300 µA/6 mA, 50 pF Drive: 0 µA/0 µA (connected to /RESET, pin C-12)

(continued)

■ **Table 13-1** Macintosh SE 68000 Direct Slot signals, loading or driving limits (continued)

Connector		Signal	Signal	Input or	Loading or driving
Row	Pin	name	description	output	limits (high/low)
B	13–17	+5V	+5 volts	Output	Drive: 1.5 A total, from all +5V pins (See the section "Power Consumption Guidelines for Macintosh SE PDS Expansion Cards" later in this chapter.)
B	18	/IPL0	Interrupt level 0 (VIA, SCSI.IRQ)	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF (Open collector; 3.3 kΩ pull-up)
B	19	/IPL1	Interrupt level 1 (SCC)	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF (Open collector; 3.3 kΩ pull-up)
B	20	/IPL2	Interrupt level 2 (NMI switch)	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF (Open collector; 3.3 kΩ pull-up)
B	21	/BERR	Bus error	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF (Open collector; 3.3 kΩ pull-up)
B	22	Spare	Not connected		
B	23–27	Reserved	For future Apple use; do not connect		
B	28	/EXT.DTK	Extended /DTACK (tristates main board's /DTACK)	Input	Load: 100 µA/2 mA, 50 pF (3.3 kΩ pull-up)
B	29	GND	Logic ground		
B	30	+12V	+12 volts	Output	Drive: 150 mA total, from all +12V pins
B	31	+12V	+12 volts	Output	(See the section "Power Consumption Guidelines for Macintosh SE PDS Expansion Cards" later in this chapter.)
B	32	-5V	–5 volts	Output	Drive: 100 mA

(continued)

■ **Table 13-1** Macintosh SE 68000 Direct Slot signals, loading or driving limits (continued)

Connector		Signal	Signal	Input or	Loading or driving
Row	Pin	name	description	output	limits (high/low)
C	1	/VPA	Valid peripheral address	Output	Drive: 40 µA/0.4 mA, 30 pF
C	2	/VMA	Valid memory address	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
C	3	/BR	Bus request	Input	Load: 100 µA/2 mA, 50 pF (3.3 kΩ pull-up)
C	4	/BGACK	Bus grant acknowledge	Input	Load: 100 µA/2 mA, 50 pF (3.3 kΩ pull-up)
C	5	/BG	Bus grant	Output	Drive: 40 µA/0.4 mA, 30 pF
C	6	/DTACK	Data transfer acknowledge	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF (3.3 kΩ pull-up, /EXT.DTK low, tristates main board's /DTACK)
C	7	R/W	Read/write	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 200 µA/2 mA, 50 pF)
C	8	/LDS	Lower data strobe	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/1 mA, 50 pF)
C	9	/UDS	Upper data strobe	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/1 mA, 50 pF)
C	10	/AS	Address strobe	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 200 µA/3.2 mA, 50 pF; 3.3 kΩ pull-up)
C	11	/PMCYC	Processor memory cycle	Output	Drive: 40 µA/0.4 mA, 30 pF (high during video access to RAM)
C	12	/RESET	System reset	In/Out	Load: 300 µA/6 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF (Open collector; 1 kΩ pull-up; connected to /HALT, pin B-12)
C	13	+5V	+5 volts	Output	Drive: 1.5 A total, from all +5V pins (See the section "Power Consumption Guidelines for Macintosh SE PDS Expansion Cards" later in this chapter.)

(continued)

■ **Table 13-1** Macintosh SE 68000 Direct Slot signals, loading or driving limits (continued)

Connector		Signal	Signal	Input or	Loading or driving
Row	Pin	name	description	output	limits (high/low)
C	14–29	D0–15	Data bus, bits 0–15	In/Out	Load: 250 µA/1 mA, 100 pF Drive: 40 µA/0.4 mA, 30 pF
C	30	GND	Logic ground		
C	31	Spare	Not connected		
C	32	–12V	–12 volts	Output	Drive: 100 mA

Functional description of the MC68000 signals in the Macintosh SE

Table 13-2 lists the MC68000 processor signals available at the Macintosh SE 68000 Direct Slot expansion connector and describes their functions.

■ **Table 13-2** MC68000 signal descriptions

Signal name	Description
A1–A23	Address lines.
D0–D15	Data bus.
/AS	Address strobe.
/BERR	Bus error. Generated by gate array due to SCSI access time-out. (Actually, /BERR is generated whenever /AS remains low for more than about 250 ms.)
/BG	Bus grant.
/BGACK	Bus grant acknowledge.
/BR	Bus request.
C16M	Gate array clock = 15.6672 MHz.
C8M	Microprocessor clock = 7.8336 MHz = C16M divided by 2.

(continued)

■ **Table 13-2** MC68000 signal descriptions (continued)

Signal name	Description
/DTACK	Data transfer acknowledge. In normal operation, /AS falls in S2 and the gate array supplies /DTACK in S4 of accesses to any address in the range $00 0000–$DF FFFF. If /AS falls after S3, /DTACK is supplied in S0 of the next access cycle (except for RAM accesses, which wait until S4 of the next cycle). /DTACK may be held off to wait for DRQ (DMA request from SCSI) in pseudo-DMA-mode SCSI accesses, to separate two successive accesses to the SCC, or to wait for a RAM access by video. /DTACK is not supplied for accesses to /VPA address space ($E0 0000–$FF FFFF). Gate array generation of /DTACK can be suppressed (put into a high-impedance state) by pulling the /EXT.DTK line low; this allows for extended generation of the /DTACK signal by an expansion card.
E	E (enable) clock.
/EXT.DTK	Pulled low to put the gate array's /DTACK output into a high-impedance state. The expansion card is then responsible for generating the /DTACK signal (as an output to the microprocessor, through the /DTACK signal line).
FC0–FC2	Function code lines.
/HALT	Halt. Wired directly to /RESET.
/IPL0–/IPL2	Interrupt priority level lines.
/LDS	Lower data strobe.
/PMCYC	Processor memory cycle. Used to synchronize with the gate array for RAM accesses. /PMCYC is low when RAM is available for microprocessor accesses and is high during video accesses. /PMCYC is always high during S0. See timing diagram, Figure 13-2.
/RESET	Reset. Wired directly to /HALT.
R/W	Read/write.
/UDS	Upper data strobe.
/VMA	Valid memory address.
/VPA	Valid peripheral address. Supplied by the gate array, coincident with /AS, for any access to VPA space ($E0 0000–$FF FFFF).

Accessing the Macintosh SE electronics from an expansion card

An expansion card slave or a peripheral I/O device simply occupies an available spot in the computer's address space, and the computer can then access the card just as it accesses any of its own I/O devices. See Figure 13-4, later in this chapter, for the Macintosh SE address space. The microprocessor on an expansion card (a coprocessor) has a more complex task than the microprocessor on the main logic board. Of course, the coprocessor can do its own work indefinitely, while the MC68000 continues to function normally, provided the expansion card's electronics are sufficiently isolated from the computer electronics. For meaningful results, however, most expansion card coprocessors will eventually need to access the I/O devices and RAM on the main logic board. To do this, the coprocessor requests the bus from the MC68000 (using /BR), the MC68000 grants the request (using /BG) and tristates itself off the bus at the end of that bus cycle, and the coprocessor then takes over as bus master (using /BGACK). At this point, the expansion card's coprocessor has complete access to all of the computer electronics.

Accessing I/O devices from an expansion card

For most Macintosh I/O devices, the timing of an access is managed entirely by the coprocessor. The coprocessor puts the device's address on the address bus and issues address strobe (/AS). For devices in the address range $00 0000 through $DF FFFF, the custom gate array, the Bob Bailey Unit, responds by selecting the correct device and issuing /DTACK. If you, the card designer, need to supply a different /DTACK on that line, the gate array's /DTACK output can be put in tristate by pulling the /EXT.DTK line low.

When a device (the VIA, for example) is accessed in the range $E0 0000 through $FF FFFF, the BBU supplies /VPA instead of /DTACK. In normal operation, the MC68000 on the Macintosh SE logic board then responds to /VPA by providing the VIA chip select /VMA, timed correctly to the E clock. After removing itself from the bus by tristate control, however, the MC68000 continues to generate its E clock but no longer provides /VMA. This means an expansion card coprocessor must correctly synchronize its VIA selection (using /VMA) and VIA accesses to the timing of the MC68000 E clock. The coprocessor can accept /VPA as its /DTACK, or provide its own.

Accessing RAM from an expansion card

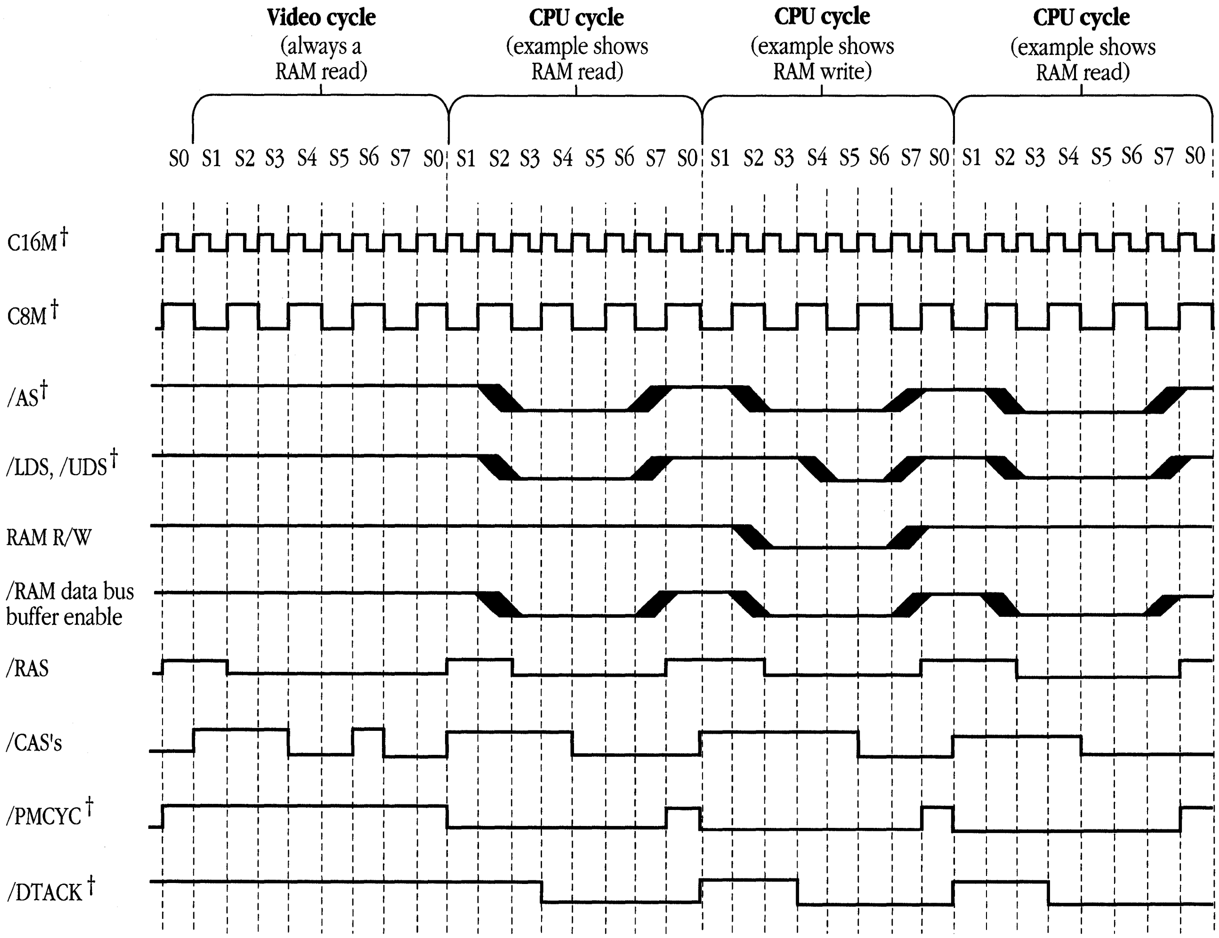
When an expansion card coprocessor accesses the RAM on the Macintosh SE logic board, the timing of these accesses is tightly constrained compared with accessing Macintosh I/O devices. Even if an expansion card coprocessor has its own on-card RAM, it usually needs to access the Macintosh SE RAM at least to update the information on the screen. This activity is always necessary because the information displayed on the Macintosh screen is always taken from the Macintosh RAM, regardless of any other RAM in the system.

As the designer of an expansion card, you may wish to synchronize the card's Macintosh RAM accesses (using /PMCYC) to avoid contention with the RAM accesses by Macintosh video circuitry. During the active portion of a screen scan line, the video uses one out of every four possible RAM accesses. These video accesses come at certain fixed times, regardless of any other activity in the system such as an expansion card coprocessor taking over the bus, or accesses to any I/O device or to the RAM itself. See Figure 13-2 for the timing of video versus processor accesses. If a coprocessor begins an access to Macintosh RAM during a video access, the coprocessor access is simply held off (/DTACK is not provided) until the following RAM access time.

Furthermore, a coprocessor must synchronize its accesses to the state machine in the BBU. This gate array is designed to generate all of the RAM control signals at the correct times. The following information will help you synchronize an expansion card coprocessor to the RAM electronics on the Macintosh SE logic board.

The BBU operates with an internal state machine that generates 16 states (S0 to SF, numbered in hex), clocked by C16M. This state machine comes up randomly, and then counts through the 16 states continuously. The state counter is not affected by anything else in the system.

■ **Figure 13-2** Timing of video and MC68000 accesses to RAM in the Macintosh SE



† This signal is available at the 96-pin expansion card connector.

There are two types of basic RAM access cycles: video/sound cycles and processor (CPU) cycles. Either type of RAM access cycle occupies eight state machine states. When video/sound cycles occur, they are always in states S8 to SF, whereas processor cycles can be either in states S0 to S7 or in states S8 to SF. To simplify discussion, however, the eight states of any RAM access cycle are numbered S0 to S7. See Figure 13-2.

A video/sound cycle occurs as a result of specific counts of the video counter. A video cycle reads two words of data from the video buffer in RAM into the gate array's Video Shift register. A sound cycle is similar to a video cycle, except that a single word from the sound/disk-speed buffer in RAM is loaded into the gate array's sound and disk-speed counters.

A processor on the main logic board or on an expansion card may access RAM during a processor cycle. A processor cycle can take place whenever a video/sound cycle is not occurring. If a processor initiates a RAM access during a video/sound cycle, the processor's RAM access is held off (/DTACK is not generated) until the video/sound cycle is complete. A processor can access devices other than RAM at any time, even during video/sound cycles.

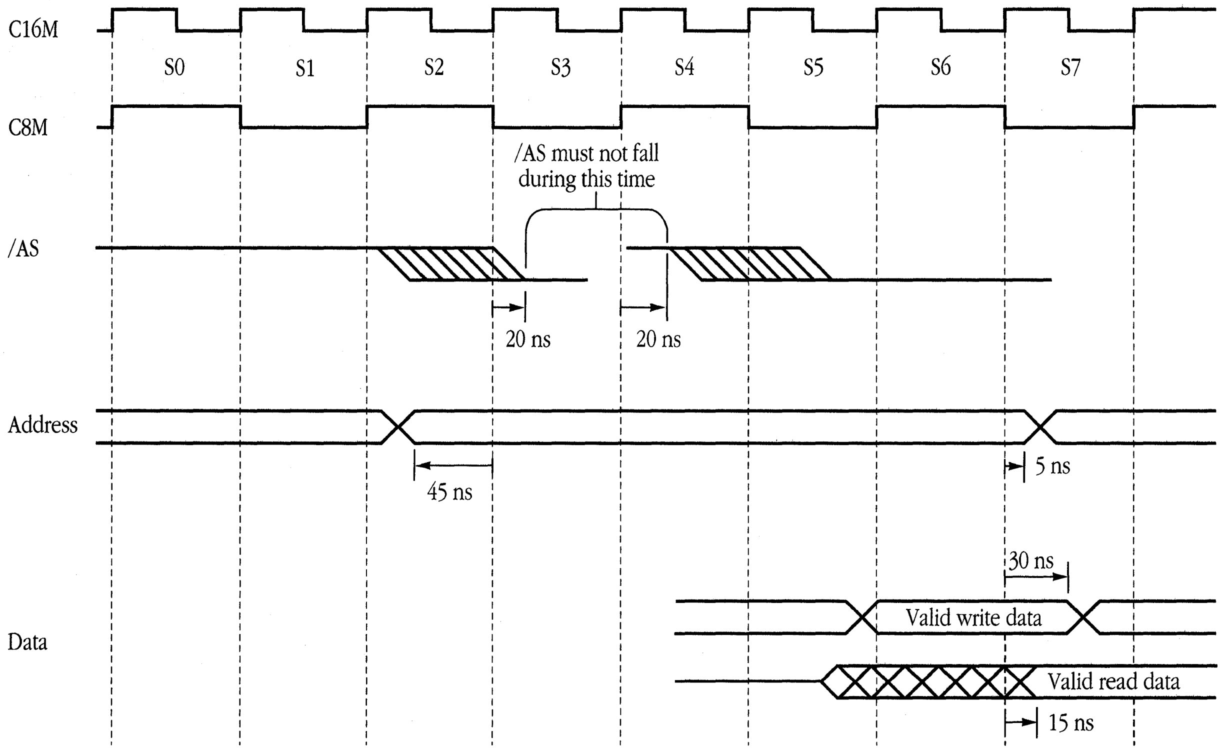
The BBU requires that a processor must not begin RAM accesses at random times. In normal operation, it expects any processor to behave more or less like an MC68000, which asserts /AS in S2 (see Figure 13-3 for details). The MC68000 in the Macintosh SE is automatically synchronized to the state machine in the BBU by the processor's receipt of /DTACK, which the gate array always asserts in S4. An expansion card can synchronize itself to the state machine in the BBU by monitoring the signal /PMCYC. See Figure 13-2 for the operation of /PMCYC. /PMCYC always falls in S1 of an eight-state processor cycle. A falling-edge detector triggered by C16M can be used to find the falling edge of /PMCYC, and therefore S1.

Pertinent timing requirements from Figure 13-3 are as follows:

- Minimum address setup time before /AS (address strobe) falls is 15 ns.
- Minimum address setup time to start of S3 is 45 ns unless /AS falls after start of S3, in which case the minimum address setup time to /AS is 45 ns.
- Address must remain valid through the first 5 ns of S7.
- /AS falling must occur not later than 20 ns into S3. If /AS has not fallen by that time, /AS must not fall until after the first 20 ns of S4 (data will be read or written in the next RAM access).
- /DTACK rises 25 ns, maximum, following start of an odd S state after /AS rises.
- Write data to the RAM must be valid from the start of S6 through the first 30 ns of S7 (when /CAS falls).
- Read data from the RAM will be valid from 15 ns into S7 until /CAS rises at the end of S0, or until /AS rises, whichever occurs first.

◆ *Note:* Clock C8M is shown only for its relation to the MC68000 state sequence. Actually, C8M is delayed relative to C16M by up to 30 ns.

■ **Figure 13-3** Timing for reading and writing RAM from a Macintosh SE expansion card



Deviating from the normal RAM access method

The coprocessor on an expansion card should operate very much like the MC68000 of the Macintosh SE when accessing the Macintosh SE RAM. In normal operation, therefore, an expansion card presents its addresses in S1, asserts /AS in S2, and receives /DTACK in S4. The following information is presented only for those designers who want to know, for some reason, exactly how far they may deviate from this normal method of operation.

To speed up RAM access, the Macintosh SE gate array internally generates /RAS if it decodes a RAM-space address anytime during S2 without waiting for /AS to indicate that the address is valid. Then, if /AS falls before the end of S3 and a RAM-space address is still present, /RAS is generated.

However, the RAM-address multiplexers switch from row addresses to column addresses at the beginning of S4, regardless of when /RAS occurs. If /AS falls later than the first 20 ns of S3, the RAM addresses will change too soon after /RAS, causing RAM errors.

Furthermore, if /AS has not fallen by the end of S3, /RAS is negated, a process that takes the first 20 ns of S4. If /AS falls during that 20 ns, a /RAS spike is generated that can cause RAM errors.

These restrictions mean that to avoid problems when addressing the Macintosh SE RAM, expansion card logic must never let /AS fall during the period from 20 ns into S3 through 20 ns into S4. See Figure 13-3. There is one exception to this: If the gate array did not decode a RAM-space address (even on a floating address bus) during S2 or the first 20 ns of S3, then /RAS is not generated, and a RAM-space address decode and /AS any time after the first 20 ns of S3 will not cause a /RAS until the usual point in the next RAM access cycle.

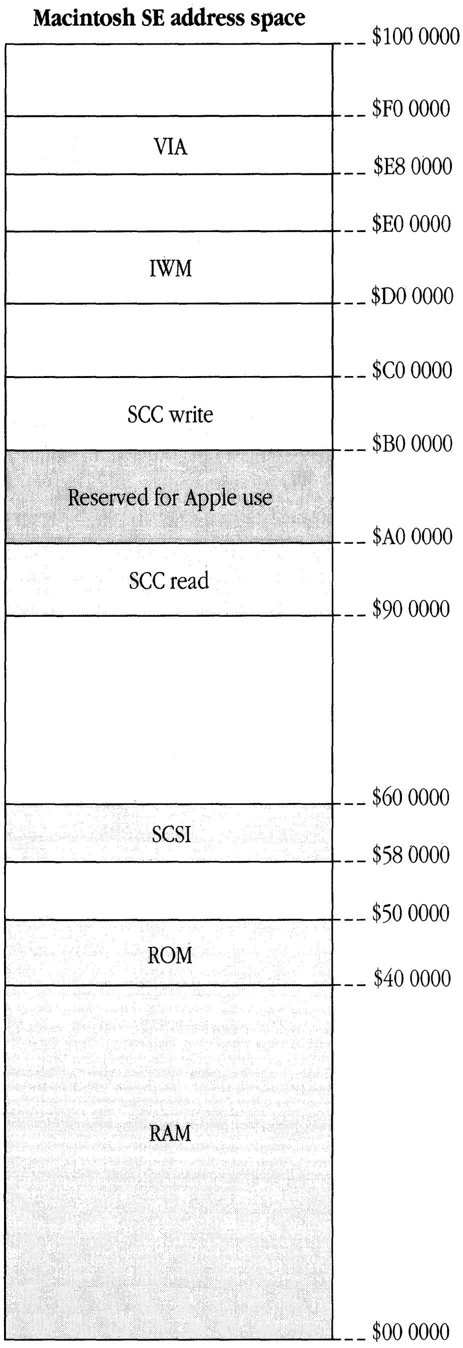
The state machine in the gate array is synchronized to the 15.6672 MHz clock, C16M, from which C8M is derived with a time delay of up to 30 ns. The MC68000 only issues /AS during even-numbered states and is synchronized to the 7.8336 MHz clock, C8M. This difference in timing sources assures that /AS in the Macintosh SE will not occur in the prohibited time interval.

Available Macintosh SE address space

The Macintosh SE address map in Figure 13-4 labels which portions of the total address space are currently used by the Macintosh SE hardware (shaded regions). Any address space not used by the Macintosh SE hardware is available for use by an expansion card. There are, of course, some limitations to this:

- For any access to the address space $00 0000 through $DF FFFF, the Macintosh SE gate array returns /DTACK in S4, following an address strobe (/AS) in S2. If /AS falls after S3, /DTACK is supplied in S0 of the next access cycle (except for RAM accesses, which wait until S4 of the next cycle). This space is best for fast, asynchronous exchanges.
- For an access to the space $E0 0000 through $FF FFFF, the gate array returns /VPA immediately following /AS, and the MC68000 then provides /VMA timed by the E clock. This space is designed for accessing slower 6800-style synchronous peripherals.
- The Macintosh SE RAM, or multiple images of that RAM, always occupy the entire address space $00 0000 through $3F FFFF.
- The Macintosh SE does not support the connection of more than one expansion card or device, so no means are provided for arbitrating among multiple external processors, or among cards that use the same address space.
- When a Macintosh SE main logic board is sent to an Apple service center for repair, Apple's board-testing equipment runs test software in address ranges $50 0000 through $51 FFFF and $F8 0000 through $F9 FFFF. Normally, those spaces may be used by an expansion card, as any such card would be removed prior to testing at an Apple service center. However, if a developer expects that customers will leave an expansion card connected to the Macintosh SE logic board when that board is sent to Apple service, such an expansion card should not use the Apple test software spaces.
- When servicing an interrupt, the MC68000 reads an address in the range $FF FFF0 through $FF FFFF. The Macintosh SE gate array returns /VPA, causing the processor to ignore any data read and to jump to the appropriate autovector location in low memory. The processor does an autovector jump only if it reads the address in servicing an interrupt, so this space may be used by an expansion card device if that device will not be confused by autovector reads.

■ **Figure 13-4** Macintosh SE address space



Power consumption guidelines for Macintosh SE PDS expansion cards

The Macintosh SE power supply supports the addition of optional Apple Desktop Bus devices, an internal hard disk, and an expansion card. Table 13-3 gives the power budget for these additions.

■ **Table 13-3** Macintosh SE 68000 Direct Slot power budget

	Amps			
Macintosh SE device	**At +5 V**	**At –5 V**	**At +12 V**	**At –12 V**
All Apple Desktop Bus devices	0.5	—	—	—
Expansion card†	1.5	0.1	0.15	0.1
Internal SCSI hard disk	1.5	—	0.9	—

† This is the allotted current for the expansion card, but, for thermal considerations, the total power of the expansion card should not exceed 7 W.

The power budget specification for the 96-pin connector allows 1.5 A to be used from all +5V pins combined. This limit is to control the heat dissipated in the restricted space over the Macintosh SE logic board, where an expansion card would be located. An additional 750 mA can be used for powering a peripheral device that is located outside of the Macintosh SE case.

The power budget specification for the 96-pin connector allows 0.15 A to be used from all +12V pins combined. Peak surge current up to 1.5 A can be tolerated briefly (up to 10 seconds)—when starting up a disk drive, for example.

68000 Direct Slot expansion for the Macintosh Portable

This section describes the electrical characteristics of the 96-pin 68000 Direct Slot expansion connector used on the Macintosh Portable. Physical guidelines for designing a Macintosh Portable 68000 Direct Slot expansion card are provided in Chapter 17. In addition to its processor-direct slot, the Macintosh Portable contains connectors for ROM expansion cards and RAM expansion cards. The RAM and ROM expansion capabilities are described in Chapter 20 and Chapter 21.

△ **Important** Before designing an expansion card for the Macintosh Portable 68000 Direct Slot, there are certain limitations that you should be aware of. First, although DC voltage and ground are available at the connector, the Macintosh Portable power budget allots no current for an expansion card. You should provide your own power supply to the expansion card. Second, in order to comply with FCC regulations on radio-frequency emissions, no connector or cable attached to an expansion card can penetrate the case of the Macintosh Portable. Finally, the small size of the card limits the size and number of components, thus severely restricting the number of features and capabilities that you can have in your design. △

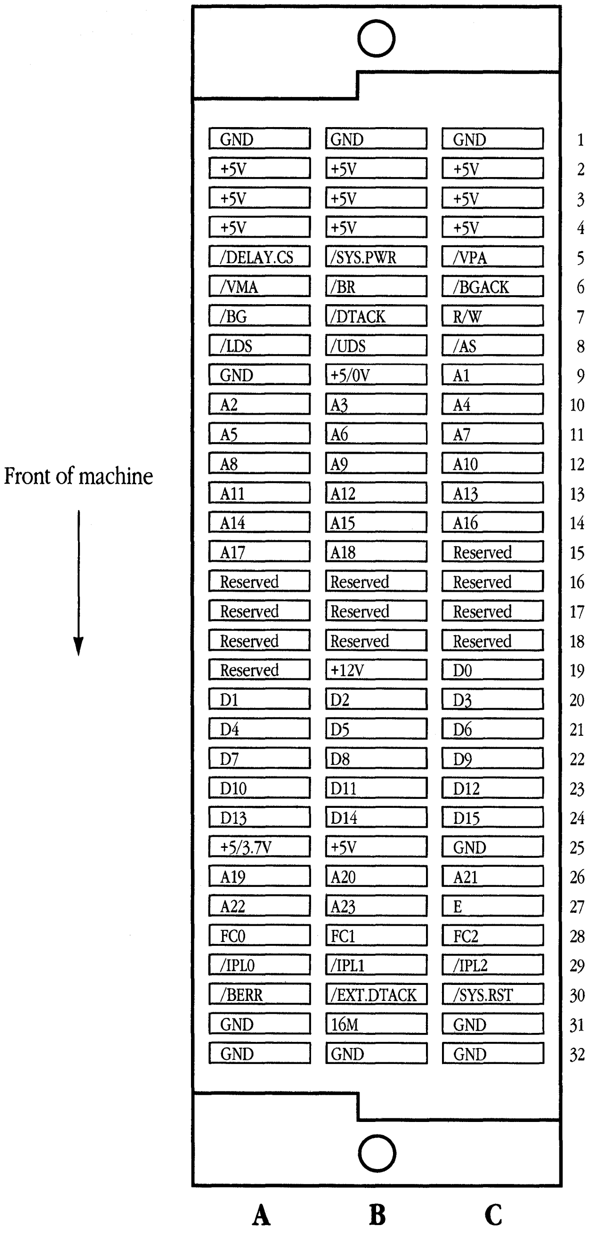
If you are determined to design an expansion card for the Macintosh Portable Direct Slot and can live within the aforementioned design limitations, you can contact Apple Macintosh Developer Technical Support (MacDTS) for additional guidance.

Remember, an expansion card designed for the Macintosh SE will not physically fit in the Macintosh Portable and vice versa.

Electrical description of the Macintosh Portable expansion connector

The Macintosh Portable expansion connector provides access to the same microprocessor signals as the Macintosh SE, but the pinout of the expansion connector is different. Figure 13-5 gives the pinout of the 96-pin expansion connector (socket) on the Macintosh Portable main logic board.

■ **Figure 13-5** Macintosh Portable 68000 Direct Slot connector pinout



	A	B	C
1	GND	GND	GND
2	+5V	+5V	+5V
3	+5V	+5V	+5V
4	+5V	+5V	+5V
5	/DELAY.CS	/SYS.PWR	/VPA
6	/VMA	/BR	/BGACK
7	/BG	/DTACK	R/W
8	/LDS	/UDS	/AS
9	GND	+5/0V	A1
10	A2	A3	A4
11	A5	A6	A7
12	A8	A9	A10
13	A11	A12	A13
14	A14	A15	A16
15	A17	A18	Reserved
16	Reserved	Reserved	Reserved
17	Reserved	Reserved	Reserved
18	Reserved	Reserved	Reserved
19	Reserved	+12V	D0
20	D1	D2	D3
21	D4	D5	D6
22	D7	D8	D9
23	D10	D11	D12
24	D13	D14	D15
25	+5/3.7V	+5V	GND
26	A19	A20	A21
27	A22	A23	E
28	FC0	FC1	FC2
29	/IPL0	/IPL1	/IPL2
30	/BERR	/EXT.DTACK	/SYS.RST
31	GND	16M	GND
32	GND	GND	GND

Functional description of the MC68HC000 signals in the Macintosh Portable

Table 13-4 lists the MC68HC000 processor signals available at the Macintosh Portable 68000 Direct Slot expansion connector and describes their functions. Notice that most of the signals are the same as the Macintosh SE processor signals.

■ **Table 13-4** MC68HC000 signal descriptions

Signal name	Description
A1–A23	Unbuffered address bus, bits 1 through 23.
D0–D15	Unbuffered data bus, bits 0 through 15.
16M	16 MHz clock.
+5/0V	+5 volts when system is active; 0 volts when system is in sleep mode.
+5/3.7V	+5 volts when system is active; +3.7 volts when system is in sleep mode.
/AS	Address strobe.
/BERR	Bus error signal, generated whenever /AS remains low for more than about 250 ms.
/BG	Bus grant.
/BGACK	Bus grant acknowledge.
/BR	Bus request.
/DELAY.CS	Input indicating that system is inserting wait states; can be used to gate chip selects.
/DTACK	Data transfer acknowledge.
E	E (enable) clock.
/EXT.DTACK	Extended data transfer acknowledge. This signal is an input to the processor logic glue that allows for extended generation of the /DTACK signal. Asserting the signal will delay generation of the /DTACK signal. The /DTACK signal will automatically be generated when the /EXT.DTACK signal is released.
FC0–FC2	Function code lines 0 through 2.
GND	Logic ground.
/IPL0–/IPL2	Input priority level lines 0 through 2.
/LDS	Lower data strobe.
Reserved	For use by Apple.
R/W	Defines bus transfer as read or write signal.

(continued)

■ **Table 13-4** MC68HC000 signal descriptions (continued)

Signal name	Description
/SYS.PWR	A signal from the Power Manager IC that causes associated circuits to tristate their outputs and go into the idle state; /SYS.PWR is pulled high (deasserted) during sleep state.
/SYS.RST	Signal that initiates a system reset.
/UDS	Upper data strobe.
/VMA	Valid memory access.
/VPA	Valid peripheral address.

Power consumption guidelines for Macintosh Portable PDS expansion cards

The power budget for the Macintosh Portable allocates a very limited amount of power to an expansion card in the processor-direct slot. The current available is given in Table 13-5. This current allocation is part of a worst-case current budget that is estimated to reduce the system operating time per battery charge by 50%.

■ **Table 13-5** Macintosh Portable 68000 Direct Slot power budget

Power supply	Operating state, mA maximum	Sleep state, mA maximum
+5 V, always on	50	1
+5 V, switched	50†	0
+12 V	25	0

† The 50 mA maximum applies to the loads of the switched and unswitched +5 V supplies.

Chapter 14 **Electrical Design Guide for 68020 Direct Slot Expansion Cards**

This chapter provides the electrical information you need to design expansion cards for Macintosh computers with a 68020 Direct Slot expansion interface. The chapter covers the following topics:

- electrical description of the 68020 Direct Slot expansion connectors for the Macintosh LC
- signal mnemonics and descriptions
- addressing issues and memory mapping
- accessing I/O and memory devices from the expansion card
- power consumption guidelines

68020 Direct Slot expansion for the Macintosh LC

A 96-pin connector on the Macintosh LC main logic board permits an expansion card to communicate directly with the 68020 processor. This feature provides an opportunity for hardware developers to increase the functionality of the Macintosh LC computer by designing expansion cards such as coprocessors, video cards, and networking cards.

This chapter gives the pinouts and describes the signal characteristics of the 68020 Direct Slot expansion connector used on the Macintosh LC. This section also provides information about the memory mapping, addressing guidelines, electrical design considerations, accessing the computer's electronics from the expansion card, and the power consumption guidelines for the expansion card in the Macintosh LC. Physical guidelines for designing a Macintosh LC Direct Slot expansion card are provided in Chapter 17.

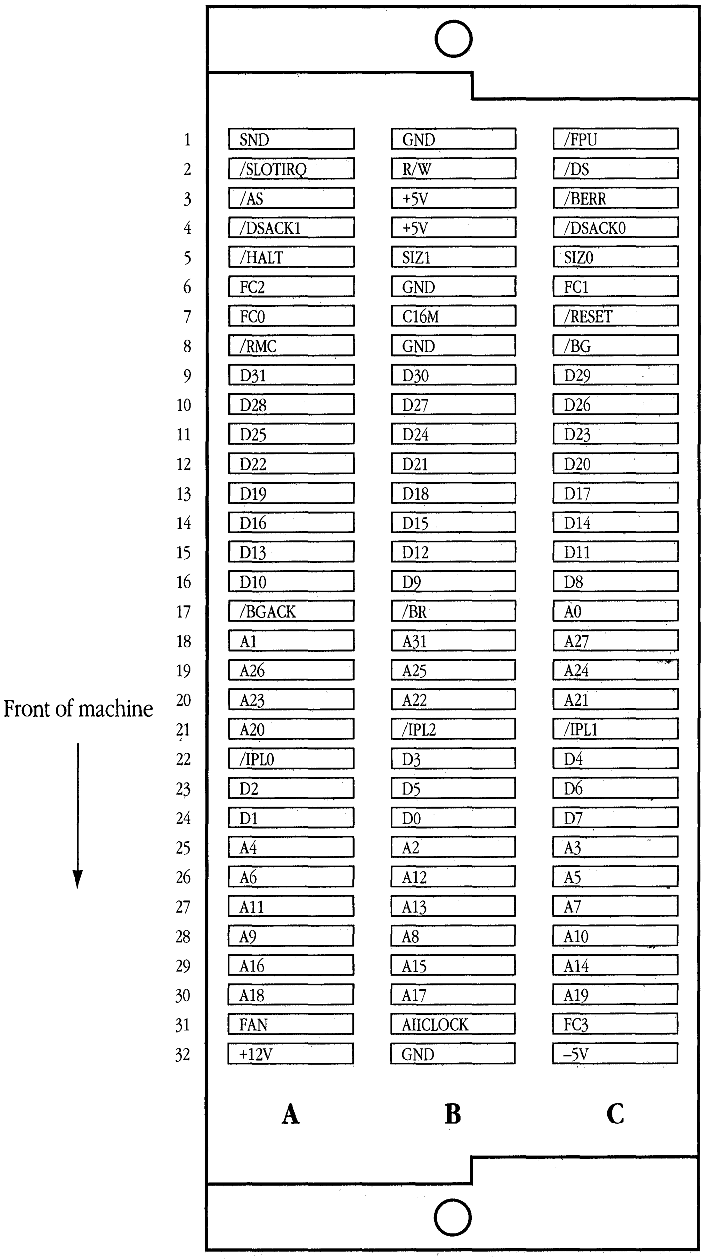
Electrical description of the expansion connector

A Euro-DIN 96-pin socket connector provides processor-direct slot (PDS) expansion for the Macintosh LC computer. This connector is physically, but not electrically, identical to the PDS connector used on the Macintosh SE. The expansion connector provides access to the MC68020 processor's full 32-bit data bus and 29 address lines, as well as to a selection of control signals.

Figure 14-1 gives the pinout for the 96-pin expansion connector (socket) on the Macintosh LC main logic board, as viewed from above.

Most of the expansion connector signals are processor direct, which means that they are tied directly to signals with identical names on the 68020 processor bus. Some of the signals do not tie directly to the processor but are used to satisfy other functional requirements of the Macintosh LC computer. Table 14-1 describes the functions of the processor-direct expansion connector signals. Table 14-2 describes the functions of non-processor-direct signals.

■ **Figure 14-1** Macintosh LC 68020 Direct Slot connector pinout



	A	B	C
1	SND	GND	/FPU
2	/SLOTIRQ	R/W	/DS
3	/AS	+5V	/BERR
4	/DSACK1	+5V	/DSACK0
5	/HALT	SIZ1	SIZ0
6	FC2	GND	FC1
7	FC0	C16M	/RESET
8	/RMC	GND	/BG
9	D31	D30	D29
10	D28	D27	D26
11	D25	D24	D23
12	D22	D21	D20
13	D19	D18	D17
14	D16	D15	D14
15	D13	D12	D11
16	D10	D9	D8
17	/BGACK	/BR	A0
18	A1	A31	A27
19	A26	A25	A24
20	A23	A22	A21
21	A20	/IPL2	/IPL1
22	/IPL0	D3	D4
23	D2	D5	D6
24	D1	D0	D7
25	A4	A2	A3
26	A6	A12	A5
27	A11	A13	A7
28	A9	A8	A10
29	A16	A15	A14
30	A18	A17	A19
31	FAN	AIICLOCK	FC3
32	+12V	GND	–5V

■ **Table 14-1** PDS expansion connector signal descriptions for the Macintosh LC

Signal name	Description
A0–A27, A31	Address lines.
D0–D31	Data lines.
/AS	Address strobe; tristate output signal indicating that valid address is on processor bus.
/BERR	Bus error; bidirectional signal indicating that invalid bus operation is being attempted.
/BG	Bus grant; output signal indicating that external device can become bus master following completion of current processor bus cycle.
/BGACK	Bus grant acknowledge; input signal indicating that external device has become bus master.
/BR	Bus request; input signal indicating that external device wishes to become bus master.
/DS	Data strobe; during read operation, /DS indicates that external device should place data on data bus; during write operation, /DS indicates that 68020 processor has placed valid data on the data bus.
/DSACK0–/DSACK1	Data transfer acknowledge signals that indicate completion of data-transfer operation.
FC0–FC2	3-bit function code used to identify address space of current bus cycle.
/HALT	Signal indicating that 68020 processor should suspend all bus activity.
/IPL0–/IPL2	Interrupt priority level lines.
/RESET	Bidirectional signal that initiates system reset.
/RMC	Tristate output signal that identifies current bus cycle as part of indivisible read-modify-write operation.
R/W	Read/write; tristate output signal that defines bus transfer as read or write operation.
SIZ0–SIZ1	Tristate output signals that work in conjunction with processor's dynamic bus sizing capabilities to indicate number of bytes remaining to be transferred during current bus cycle.

■ **Table 14-2** Non-processor-direct expansion connector signals for the Macintosh LC

Signal name	Description
AIICLOCK	Pin that requires a 17.234 MHz clock input to generate timing for 560 × 384 video mode.
C16M	15.6672 MHz clock that runs CPU.
FAN	Voltage, normally about 2 V, required to operate Macintosh LC fan. If your expansion card requires more cooling, you can ground this pin to speed up the fan.
/FC3	Function code bit that selects memory address map. Low selects 24-bit map; high selects 32-bit map.
/FPU	Select signal for optional floating-point unit.
/SLOTIRQ	Signal that expansion card uses to generate interrupt request. When active low, this signal generates a level-2 interrupt if slot interrupt enable bit in V8 chip is set. (Similar to the /NMRQ line in Macintosh computers with NuBus.)
SND	Input to speaker amplifier that permits you to drive speaker from expansion card without involving CPU.

Load/drive limits of the PDS expansion connector signals for the Macintosh LC

Table 14-3 provides the load presented or drive available to each pin of an expansion card and indicates whether the signals are inputs or outputs. The load values in Table 14-3 are based on real load after incorporating the existing loads on the main logic board.

In the column labeled "Input/Output" in Table 14-3, input refers to a signal from the expansion card to the processor and corresponds directly to the load shown in the column labeled "Load or Drive Limits." Output refers to a signal from the processor to the expansion card and corresponds directly to the drive shown in that column. An example may be helpful in interpreting the "Load or Drive Limits" column. The /RESET line is shown as presenting a load of 100 µA/8 mA, 200 pF. This is the maximum expected load that an expansion card must drive when sending a /RESET signal to the main logic board. The DC load is given in the format *signal high/signal low.* This means that the expansion card must drive a load of up to 100 µA when it drives /RESET high (logic 1) and a load of up to 8 mA when it drives /RESET low (logic 0). The AC load is given as 200 pF, the maximum capacitance to ground presented by the main logic board to AC signals from an expansion card.

In addition, /RESET presents a drive of 50 µA/50 µA, 30 pF. This is the maximum amount of drive from the main logic board that is available to integrated circuits on the expansion card. The /RESET line can drive an expansion card DC load of up to 50 µA in both the high state and the low state. The AC drive is given as 30 pF, the maximum capacitance to ground that an expansion card may present to AC signals from the /RESET line.

■ **Table 14-3** Macintosh LC 68020 Direct Slot signals, loading or driving limits

Signal name	Input/output	Load or drive limits
A0–A31	In/Out	Load: 100 µA/8 mA, 100 pF Drive: 50 µA/400 µA, 30 pF
D0–D15	In/Out	Load: 500 µA/1 mA, 80 pF Drive: 40 µA/.2 mA, 30 pF
D16–D23	In/Out	Load: 500 µA/1 mA, 100 pF Drive: 40 µA/.2 mA, 30 pF
D24–D31	In/Out	Load: 500 µA/1 mA, 130 pF Drive: 40 µA/.2 mA, 30 pF
AIICLOCK	Input	Load: 100 µA/8 mA, 20 pF 4.7 kΩ pull-up
/AS	In/Out	Load: 100 µA/8 mA, 75 pF Drive: 40 µA/2 mA, 30 pF 4.7 kΩ pull-up
/BERR	In/Out	Load: 100 µA/8 mA, 75 pF Drive: 100 µA/1 mA, 30 pF 4.7 kΩ pull-up
/BG	Output	Drive: 40 µA/400 µA, 30 pF
/BGACK	Input	Load: 100 µA/8 mA, 75 pF 4.7 kΩ pull-up
/BR	Input	Load: 100 µA/8 mA, 75 pF 4.7 kΩ pull-up
C16M	Output	Drive: 100 µA/100 µA, 20 pF
/DS	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF
/DSACK0–/DSACK1	In/Out	Load: 100 µA/8 mA, 75 pF Drive: 100 µA/1 mA, 30 pF 4.7 kΩ pull-up
FAN	Input	Pin that increases speed of the fan and improves thermal conditions when digitally grounded

(continued)

■ **Table 14-3** Macintosh LC 68020 Direct Slot signals, loading or driving limits (continued)

Signal name	Input/output	Load or drive limits
FC0–FC2	In/Out	Load: 100 μA/8 mA, 75 pF Drive: 100 μA/1 mA, 30 pF
/FC3	Output	Drive: 1 mA/1 mA, 20 pF
/FPU	Output	Drive: 100 μA/100 μA, 30 pF
GND (analog only)	In/Out	The audio subsystem ground; not to be used for digital ground
/HALT	In/Out	Load: 100 μA/8 mA, 75 pF Drive: 100 μA/5 mA, 50 pF 4.7 kΩ pull-up
/IPL0–/IPL2	Output	Drive: 40 μA/400 μA, 30 pF 4.7 kΩ pull-up
/RESET	In/Out	Load: 100 μA/8 mA, 200 pF Drive: 50 μA/50 μA, 30 pF
/RMC	Output	Drive: 100 μA/2 mA, 30 pF
R/W	In/Out	Load: 100 μA/8 mA, 100 pF Drive: 40 μA/1 mA, 30 pF
SIZ0–SIZ1	In/Out	Load: 100 μA/100 μA, 75 pF Drive: 100 μA/1 mA, 30 pF
/SLOTIRQ	Input	Load: 100 μA/8 mA, 20 pF 4.7 kΩ pull-up
SND	Input	Use transistor with analog grounded emitter and series resistor (4.7 kΩ nominal) 4.7 kΩ pull-up

△ **Important** Under no circumstances should you use the analog GND pin (Row B, pin 1) for a digital ground on an expansion card. Doing so will cause digital noise to be coupled into the audio system, resulting in degraded sound quality. △

Electrical design guidelines for the Macintosh LC 68020 Direct Slot expansion card

This section provides the electrical information you need to design an expansion card for the Macintosh LC computer.

Address decode and memory mapping

The Macintosh LC uses a new custom chip, the V8 gate array. One of the V8 gate array's most important functions is to provide address decode and memory-mapping functions for the Macintosh LC. The V8 implements two memory address-mapping modes, a 24-bit mode and a 32-bit mode. Table 14-4 lists the memory map in each mode. As in other Macintosh computers, a control bit determines which map is to be used.

Notice that in the 24-bit mode, the highest address byte (8 bits) is not used, and all addressable devices appear in the 16 MB address space.

■ **Table 14-4** Macintosh LC memory map summary

Function	24-bit mode	32-bit mode
RAM	$00 0000–$9F FFFF	$0000 0000–$009F FFFF
ROM	$A0 0000–$DF FFFF	$40A0 0000–$40DF FFFF
Expansion space	$E0 0000–$EF FFFF	$FE00 0000–$FEFF FFFF
I/O space	$F0 0000–$FF FFFF	$50F0 0000–$50FF FFFF
VRAM	$FC 0000–$FF FFFF	$50FC 0000–$50FF FFFF

◆ *Note:* The Macintosh LC computer is shipped with 256 or 512 KB of VRAM installed in a SIMM socket. With this configuration, main memory is not used for storing video data. If VRAM is not installed, it is possible to use main memory for video storage but only to support the 640 × 480 monochrome video mode. This practice is not recommended.

A bus error results if you try to access address $FF FFFF in the 24-bit mode or address $FFFF FFFF in the 32-bit mode.

Addressing guidelines

Although the Macintosh LC computer does not have the NuBus expansion interface, you should design your expansion card to occupy an address location corresponding to the 32-bit address space that would be occupied by a NuBus card in slot space $E. This method of emulating NuBus address space is called **pseudoslot design.** The only additional constraints to your design are a need for a declaration ROM and adherence to some address-decoding rules.

The expansion card address is a function of the memory map selected. The card appears in address space $E0 0000 through $EF FFFF in the 24-bit map and in address space $FE00 0000 through $FFFF FFFF in the 32-bit map. However, to allow the existing Slot Manager to control your card as though it were a NuBus card in slot $E, software must address the card as if it were in either the 16 MB standard slot space ($FE00 0000 through $FEFF FFFF) or the 256 MB super slot space ($E000 0000 through $EFFF FFFF). This means that you will not have to develop a new software driver because the driver for the NuBus expansion interface will also work with your processor-direct expansion card.

To ensure compatibility with future hardware and software, you should decode all the address bits to minimize the chance of address conflicts. To ensure that the Slot Manager recognizes your card, make sure that the declaration ROM resides at the upper address limit of the 16 MB address space. To find out more about the Slot Manager, refer to Chapter 8, "NuBus Card Firmware," in this book and the Slot Manager information in *Inside Macintosh.*

Electrical design considerations

When designing an expansion card for the Macintosh LC computer, you should make sure that your card's timing matches the timing requirements of the MC68020 microprocessor. For information on the processor's timing requirements, see the *MC68020 32-Bit Microprocessor User's Manual.*

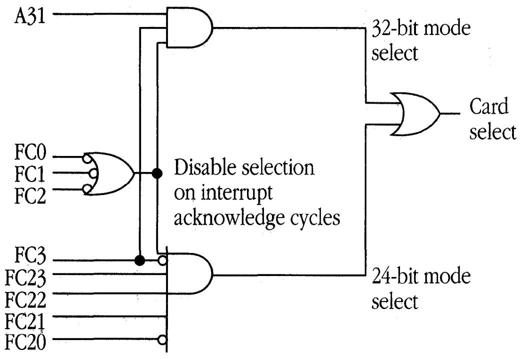
To protect the timing signal margins of the Macintosh LC computer's main logic board, your design should never extend the processor signals beyond their specified current load limits.

Since most of the expansion connector signals connect to MOS (metal oxide semiconductor) devices, the DC load on the processor bus signals is minimal. Only one LS (low-power Schottky) load is connected to the high 16 data lines (D31 to D16). All of the signals can drive at least one TTL (transistor-transistor logic) load (1.6 mA sink current and 400 µA source current).

◆ *Note:* It is recommended that all devices that attach to the LC data bus signals be MOS or LS devices to minimize the DC load. Do not pull up or pull down resistors on the data bus.

Your expansion card must generate its own card select signal. Figure 14-2 shows a typical example of the required logic. Notice that a function code of 111 (CPU space) disables the card select signal. This action is important because it prevents the card from being selected during interrupt acknowledge cycles.

■ **Figure 14-2** Macintosh LC expansion card selection logic



A bus timer in the Macintosh LC computer generates a bus error (/BERR) signal if the expansion card fails to transmit a data transfer acknowledge (/DSACK) signal within 45 to 90 µs of the active low assertion of the address strobe (/AS) signal. The /BERR signal is generated after counting two clock cycles while /AS is low. Your expansion card design must generate a /DSACK or other termination signal within this period. This action permits the Slot Manager to determine whether an expansion card is in the slot. Since, in the 32-bit mode of operation, address bit A31 = 1 selects the expansion card, it is not necessary for the Slot Manager to search all NuBus slots for cards as it does in other Macintosh systems; only slot space $E will be scanned.

Accessing I/O and memory devices from the Macintosh LC expansion card

The expansion card's task of accessing resources on the main logic board is somewhat more complex than the 68020 processor's task of accessing the electronics on an expansion card. When the expansion card needs access to the Macintosh LC computer's resources, it activates the bus request (/BR) signal to request the bus from the 68020 processor. The processor grants the bus (/BG) and tristates itself off the bus at the end of that bus cycle. The processor also generates the bus grant acknowledge (/BGACK) signal to indicate that an external device has become bus master. At this point, the expansion card coprocessor has complete access to all of the Macintosh LC electronics.

The timing of an access is controlled by the V8 gate array for all devices on the main logic board. Once the coprocessor is bus master, it asserts a valid address and an address strobe (/AS) signal. The gate array detects the address, generates the necessary chip selects, and activates the data transfer acknowledge (/DSACK) signals to inform the coprocessor of cycle completion.

Power consumption guidelines for Macintosh LC PDS expansion cards

The power budget for the Macintosh LC allocates a limited amount of power to an expansion card in the processor-direct slot. The current available is given in Table 14-5.

■ **Table 14-5** Macintosh LC 68020 Direct Slot power budget

Voltage, V	Current load, mA
+5	800
–5	20
+12	200

Power restrictions in the Macintosh LC computer limit the amount of power that can be dissipated by an expansion card to a maximum of 4 W. The entire 4 W can be from the +5V supply, or from a combination of the three supply voltages, but the total shall not exceed 4 W.

▲ **Warning** Cards dissipating more than 4 W may overheat and damage the Macintosh LC computer's circuitry or cause it to become inoperable. ▲

Chapter 15 **Electrical Design Guide for 68030 Direct Slot Expansion Cards**

This chapter provides electrical guidelines for designing PDS expansion cards for the Macintosh SE/30, the Macintosh IIsi, and the Macintosh IIfx. This section includes information on the following topics:

- electrical implementation of the 68030 Direct Slot
- functional description of expansion connector signals
- accessing the main logic board electronics from an expansion card
- accessing I/O and memory devices from an expansion card
- pseudoslot expansion card design guidelines
- power consumption guidelines
- the Macintosh IIsi PDS adapter card

68030 Direct Slot expansion

The 68030 Direct Slot is used on compact, non-NuBus computers such as the Macintosh SE/30, but is also used on modular machines such as the Macintosh IIsi and the Macintosh IIfx that have both NuBus and PDS interfaces. To support the 32-bit address and data buses of the MC68030 microprocessor, the pin count of the 68030 Direct Slot expansion connector was increased to 120 pins, as opposed to the 96-pin connector used on Macintosh MC68000-based and MC68020-based machines.

The following sections discuss compatibility issues, describe the pin assignments, define the signals, and provide signal load and drive information for the implementation of the 68030 Direct Slot on the Macintosh SE/30, the Macintosh IIsi, and the Macintosh IIfx computers. This information is followed by two more sections that give specific design guidelines for the Macintosh SE/30 expansion cards, the Macintosh IIsi expansion cards, and the Macintosh IIfx expansion cards, respectively.

68030 Direct Slot expansion card compatibility issues

Although the PDS expansion connectors on the Macintosh SE/30, Macintosh IIsi, and Macintosh IIfx are physically the same, there are some electrical differences between the connectors. The pinouts of the expansion connectors used on the Macintosh SE/30 and the Macintosh IIfx are nearly identical except for certain signals that are machine-specific (unique) to each computer. Although Apple has made every attempt to make any differences between the two connectors transparent to developers, expansion cards designed for the Macintosh SE/30 and the Macintosh IIfx computers are not interchangeable.

The Macintosh IIsi differs further from the Macintosh SE/30 and the Macintosh IIfx. To install a PDS expansion card in the Macintosh IIsi, you must first install a PDS adapter card. An MC68030 Direct Slot adapter kit is available from an authorized Apple dealer. This expansion scheme is similar to the NuBus expansion scheme for the Macintosh IIsi. Once the PDS adapter card has been installed in the Macintosh IIsi, the expansion connector is physically and electrically identical to the one found on the Macintosh SE/30.

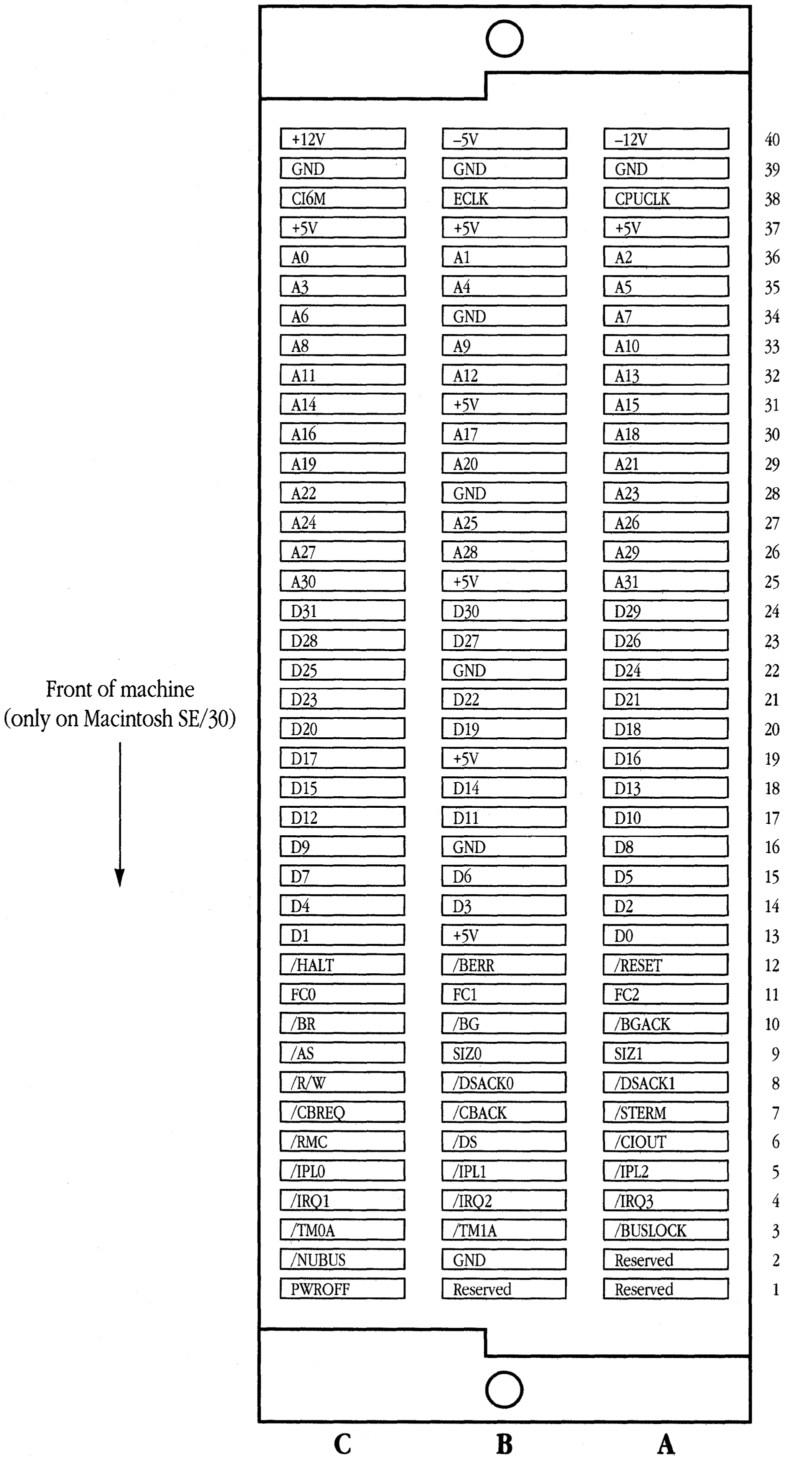
However, there are two major design differences between the Macintosh SE/30 and the Macintosh IIsi. These differences may prevent PDS cards developed for one MC68030 computer from working in the other. The clock speeds of the Macintosh SE/30 and the Macintosh IIsi are different, so that a PDS expansion card developed for the Macintosh SE/30 must be able to run at the Macintosh IIsi clock speed of 20 MHz. The other difference between the Macintosh IIsi and the Macintosh SE/30 is the way that RAM accesses are structured. This difference has serious implications for MC68030 PDS master cards. The Macintosh SE/30 responds to RAM accesses with /DSACK0 and /DSACK1. The Macintosh IIsi responds only with /STERM. Because of this difference, expansion cards, especially PDS master cards, may not be interchangeable.

Electrical description of the Macintosh SE/30 and the Macintosh IIsi 68030 Direct Slot

◆ *Note:* The following description of the Macintosh IIsi assumes that you have already installed the MC68030 Direct Slot adapter kit. To get more detailed information about the expansion connector on the Macintosh IIsi main logic board in which the adapter card fits, refer to the last section in this chapter, "Macintosh IIsi PDS Adapter Card."

The PDS expansion card interfaces for the Macintosh SE/30 and the Macintosh IIsi are identical. The only differences between the two expansion card connectors are the clock speed and the load limits for each signal. Figure 15-1 gives the pinouts for the 120-pin expansion connector on both the Macintosh SE/30 main logic board and the Macintosh IIsi adapter card. Table 15-1 lists the pin assignments, gives the signal names, and briefly describes each signal.

■ **Figure 15-1** Macintosh SE/30 and Macintosh IIsi 68030 Direct Slot connector pinout



Front of machine
(only on Macintosh SE/30)

	C	B	A
40	+12V	–5V	–12V
39	GND	GND	GND
38	C16M	ECLK	CPUCLK
37	+5V	+5V	+5V
36	A0	A1	A2
35	A3	A4	A5
34	A6	GND	A7
33	A8	A9	A10
32	A11	A12	A13
31	A14	+5V	A15
30	A16	A17	A18
29	A19	A20	A21
28	A22	GND	A23
27	A24	A25	A26
26	A27	A28	A29
25	A30	+5V	A31
24	D31	D30	D29
23	D28	D27	D26
22	D25	GND	D24
21	D23	D22	D21
20	D20	D19	D18
19	D17	+5V	D16
18	D15	D14	D13
17	D12	D11	D10
16	D9	GND	D8
15	D7	D6	D5
14	D4	D3	D2
13	D1	+5V	D0
12	/HALT	/BERR	/RESET
11	FC0	FC1	FC2
10	/BR	/BG	/BGACK
9	/AS	SIZ0	SIZ1
8	/R/W	/DSACK0	/DSACK1
7	/CBREQ	/CBACK	/STERM
6	/RMC	/DS	/CIOUT
5	/IPL0	/IPL1	/IPL2
4	/IRQ1	/IRQ2	/IRQ3
3	/TM0A	/TM1A	/BUSLOCK
2	/NUBUS	GND	Reserved
1	PWROFF	Reserved	Reserved

■ **Table 15-1** Macintosh SE/30 and Macintosh IIsi 68030 Direct Slot connector signals

Connector Row	 Pin	Signal name	Signal description
A	1	Reserved	For use by Apple; this pin is different on the Macintosh IIsi main logic board. Apple has chosen not to extend this signal out to the PDS adapter card. See the section "Macintosh IIsi PDS Adapter Card" later in this chapter.
A	2	Reserved	For use by Apple; this pin is different on the Macintosh IIsi main logic board. Apple has chosen not to extend this signal out to the PDS adapter card. See the section "Macintosh IIsi PDS Adapter Card" later in this chapter.
A	3	/BUSLOCK	NuBus buslock
A	4	/IRQ3	Interrupt input 3
A	5	/IPL2	Interrupt priority 2
A	6	/CIOUT	Cache inhibit out
A	7	/STERM	Synchronous termination
A	8	/DSACK1	Data acknowledge 1
A	9	SIZ1	Transfer size bit 1
A	10	/BGACK	Bus grant acknowledge
A	11	FC2	Function code 2
A	12	/RESET	System reset
A	13	D0	Data bit 0
A	14	D2	Data bit 2
A	15	D5	Data bit 5
A	16	D8	Data bit 8
A	17	D10	Data bit 10
A	18	D13	Data bit 13
A	19	D16	Data bit 16
A	20	D18	Data bit 18
A	21	D21	Data bit 21
A	22	D24	Data bit 24
A	23	D26	Data bit 26
A	24	D29	Data bit 29
A	25	A31	Address bit 31
A	26	A29	Address bit 29
A	27	A26	Address bit 26
A	28	A23	Address bit 23
A	29	A21	Address bit 21
A	30	A18	Address bit 18
A	31	A15	Address bit 15

(continued)

■ **Table 15-1** Macintosh SE/30 and Macintosh IIsi 68030 Direct Slot connector signals (continued)

Connector Row	Pin	Signal name	Signal description
A	32	A13	Address bit 13
A	33	A10	Address bit 10
A	34	A7	Address bit 7
A	35	A5	Address bit 5
A	36	A2	Address bit 2
A	37	+5V	5 volts
A	38	CPUCLK	15.6672 MHz CPU clock for the Macintosh SE/30; 20 MHz CPU clock for the Macintosh IIsi
A	39	GND	Ground
A	40	–12V	–12 volts
B	1	Reserved	For use by Apple; this pin is different on the Macintosh IIsi main logic board. Apple has chosen not to extend this signal out to the PDS adapter card. See the section "Macintosh IIsi PDS Adapter Card" later in this chapter.
B	2	GND	Ground
B	3	/TM1A	NuBus transfer mode bit 1
B	4	/IRQ2	Interrupt input 2
B	5	/IPL1	Interrupt priority 1
B	6	/DS	Data strobe
B	7	/CBACK	Cache burst acknowledge
B	8	/DSACK0	Data acknowledge 0
B	9	SIZ0	Transfer size bit 0
B	10	/BG	Bus grant
B	11	FC1	Function code 1
B	12	/BERR	Bus error
B	13	+5V	5 volts
B	14	D3	Data bit 3
B	15	D6	Data bit 6
B	16	GND	Ground
B	17	D11	Data bit 11
B	18	D14	Data bit 14
B	19	+5V	5 volts
B	20	D19	Data bit 19
B	21	D22	Data bit 22
B	22	GND	Ground
B	23	D27	Data bit 27
B	24	D30	Data bit 30

(continued)

■ **Table 15-1** Macintosh SE/30 and Macintosh IIsi 68030 Direct Slot connector signals (continued)

Connector Row	Pin	Signal name	Signal description
B	25	+5V	5 volts
B	26	A28	Address bit 28
B	27	A25	Address bit 25
B	28	GND	Ground
B	29	A20	Address bit 20
B	30	A17	Address bit 17
B	31	+5V	5 volts
B	32	A12	Address bit 12
B	33	A9	Address bit 9
B	34	GND	Ground
B	35	A4	Address bit 4
B	36	A1	Address bit 1
B	37	+5V	5 volts
B	38	ECLK	E clock
B	39	GND	Ground
B	40	–5V	–5 volts
C	1	PFW†	Shutdown bit
C	2	/NUBUS	NuBus space access
C	3	/TM0A	NuBus transfer mode bit 0
C	4	/IRQ1	Interrupt input 1
C	5	/IPL0	Interrupt priority 0
C	6	/RMC	Read modify cycle
C	7	/CBREQ	Cache burst request
C	8	/R/W	Read/write
C	9	/AS	Address strobe
C	10	/BR	Bus request
C	11	FC0	Function code 0
C	12	/HALT	Halt
C	13	D1	Data bit 1
C	14	D4	Data bit 4
C	15	D7	Data bit 7
C	16	D9	Data bit 9
C	17	D12	Data bit 12
C	18	D15	Data bit 15
C	19	D17	Data bit 17
C	20	D20	Data bit 20
C	21	D23	Data bit 23

(continued)

■ **Table 15-1** Macintosh SE/30 and Macintosh IIsi 68030 Direct Slot connector signals (continued)

Connector **Row**	 **Pin**	**Signal** **name**	**Signal** **description**
C	22	D25	Data bit 25
C	23	D28	Data bit 28
C	24	D31	Data bit 31
C	25	A30	Address bit 30
C	26	A27	Address bit 27
C	27	A24	Address bit 24
C	28	A22	Address bit 22
C	29	A19	Address bit 19
C	30	A16	Address bit 16
C	31	A14	Address bit 14
C	32	A11	Address bit 11
C	33	A8	Address bit 8
C	34	A6	Address bit 6
C	35	A3	Address bit 3
C	36	A0	Address bit 0
C	37	+5V	5 volts
C	38	C16M	15.6672 MHz clock
C	39	GND	Ground
C	40	+12V	+12 volts

† On the Macintosh SE/30, this signal name has been referred to as PWROFF, but its function is the same on both the Macintosh IIsi and the Macintosh SE/30.

Because the load limits for the signals on the Macintosh IIsi and the Macintosh SE/30 differ, there are separate tables that describe each. Table 15-2 provides the load presented or drive available to each pin of an expansion card on the Macintosh SE/30 and indicates whether the signals are inputs or outputs. The load or drive limits for each pin on the Macintosh IIsi expansion card are described in Table 15-3.

In the column labeled "Input/Output" in Table 15-2 and Table 15-3, input refers to a signal from the expansion card to the processor and corresponds directly to the load shown in the column labeled "Load or Drive Limits." Output refers to a signal from the processor to the expansion card and corresponds directly to the drive shown in that column. An example may be helpful in interpreting the "Load or Drive Limits" column. The /RESET line is shown as presenting a load of 300 µA/8 mA, 50 pF. This is the maximum expected load that an expansion card must drive when sending a /RESET signal to the main logic board. The DC load is given in the format *signal high/signal low.* This means that the expansion card must drive a load of up to 300 µA when it drives /RESET high (logic 1) and a load of up to 8 mA when it drives /RESET low (logic 0). The AC load is given as 50 pF, the maximum capacitance to ground presented by the main logic board to AC signals from an expansion card. The notation "Open collector; 1 kΩ pull-up" in the table means that the /RESET line is normally in the open collector state; it is only driven low, and a 1 kΩ pull-up resistor on the main logic board returns the line to a logic 1.

In addition, /RESET presents a drive of 40 µA/0.4 mA, 30 pF. This is the maximum amount of drive from the main logic board that is available to integrated circuits on the expansion card. The /RESET line can drive an expansion card DC load of up to 40 µA in the high state or up to 0.4 mA in the low state. The AC drive is given as 30 pF, the maximum capacitance to ground that an expansion card may present to AC signals from the /RESET line.

Some of the expansion connector signals are specified to drive one 74LS input (a standard 74LS input load is 20 µA high, 0.2 mA low); other signals can drive two 74LS inputs. These strict limitations are imposed to protect the noise and timing margins of the main logic board. All signals needed by an expansion card should be buffered at the expansion connector. The use of newer logic families with very low input loading allows you more margin and flexibility in your expansion card designs.

Where "Load:" is in parentheses, the pin carries a signal that is usually an output driven by the MC68030 but that is tristated by the MC68030 after granting the bus to a DMA requester. When tristated by the MC68030, this signal may be driven by an expansion card.

■ **Table 15-2** Macintosh SE/30 68030 Direct Slot signals, loading or driving limits

Signal name	Input/output	Load or drive limits
A0–A31	In/Out	Load: 300 µA/3 mA, 100 pF Drive: 40 µA/0.4 mA, 30 pF
D0–D23	In/Out	Load: 150 µA/150 µA, 100 pF Drive: 40 µA/0.4 mA, 30 pF
D24–D31	In/Out	Load: 300 µA/300 µA, 100 pF Drive: 20 µA/0.2 mA, 30 pF
/AS	Output (Input)	Drive: 40 µA/0.2 mA, 30 pF (Load: 100 µA/8 mA, 50 pF) Open collector, 1 kΩ pull-up
/BERR	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF Open collector, 1 kΩ pull-up
/BG	Output	Drive: 40 µA/0.4 mA, 30 pF
/BGACK	Input	Load: 100 µA/8 mA, 50 pF 1 kΩ pull-up
/BR	Input	Load: 100 µA/8 mA, 50 pF 1 kΩ pull-up
/BUSLOCK	Input	Load: 400 µA/2 mA, 50 pF
C16M	Output	Drive: 40 µA/0.4 mA, 30 pF
/CBACK	Input	Load: 100 µA/100 µA, 50 pF
/CBREQ	Output	Drive: 40 µA/0.4 mA, 30 pF
/CIOUT	Output	Drive: 40 µA/0.4 mA, 30 pF
CPUCLK	Output	Drive: 40 µA/0.4 mA, 30 pF
/DS	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF Load: 100 µA/8 mA, 50 pF Open collector, 1 kΩ pull-up
/DSACK0–/DSACK1	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/0.2 mA, 30 pF Open collector, 1 kΩ pull-up
ECLK	Output	Drive: 40 µA/0.4 mA, 30 pF
FC0–FC2	Output (Input)	Drive: 20 µA/0.2 mA, 30 pF (Load: 100 µA/8 mA, 50 pF) Open collector, 1 kΩ pull-up

(continued)

■ **Table 15-2** Macintosh SE/30 Direct Slot signals, loading or driving limits (continued)

Signal name	Input/output	Load or drive limits
/HALT	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF Open collector, 1 kΩ pull-up
/IPL0–/IPL2	In/Out	Load: 100 µA/100 µA, 50 pF Drive: 40 µA/0.4 mA, 30 pF
/IRQ0–/IRQ3	Input	Load: 400 µA/4 mA, 50 pF
/NUBUS	Output	Drive: 40 µA/0.4 mA, 30 pF
PFW	Output	Drive: 40 µA/0.4 mA, 30 pF
/RESET	In/Out	Load: 300 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF Open collector, 1 kΩ pull-up
/RMC	Output	Drive: 40 µA/0.4 mA, 30 pF
R/W	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/8 mA, 50 pF) Open collector, 1 kΩ pull-up
SIZ0–SIZ1	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
/STERM	Input	Load: 100 µA/100 µA, 50 pF
/TM0A	Input	Load: 400 µA/2 mA, 50 pF
/TM1A	Input	Load: 400 µA/2 mA, 50 pF

■ **Table 15-3** Macintosh IIsi 68030 Direct Slot signals, loading or driving limits

Signal name	Input/output	Load or drive limits
A0–A31	In/Out	Load: 300 μA/5 mA, 100 pF Drive: 40 μA/0.4 mA, 30 pF
D0–D23	In/Out	Load: 300 μA/5 mA, 100 pF Drive: 40 μA/0.4 mA, 30 pF
D24–D31	In/Out	Load: 300 μA/5 mA, 100 pF Drive: 20 μA/0.2 mA, 30 pF
/AS	Output (Input)	Drive: 40 μA/0.2 mA, 30 pF (Load: 100 μA/8 mA, 50 pF) Open collector, 1 kΩ pull-up
/BERR	In/Out	Load: 100 μA/8 mA, 50 pF Drive: 40 μA/0.4 mA, 30 pF Open collector, 1 kΩ pull-up
/BG	Output	Drive: 40 μA/0.4 mA, 30 pF
/BGACK	Input	Load: 100 μA/8 mA, 50 pF 1 kΩ pull-up
/BR	Input	Load: 100 μA/8 mA, 50 pF 1 kΩ pull-up
/BUSLOCK	Input	Load: 400 μA/2 mA, 50 pF
C16M	Output	Drive: 40 μA/0.4 mA, 30 pF
/CBACK	Input	Load: 100 μA/100 μA, 50 pF
/CBREQ	Output	Drive: 40 μA/0.4 mA, 30 pF
/CIOUT	Output	Drive: 40 μA/0.4 mA, 30 pF
CPUCLK	Output	Drive: 40 μA/0.4 mA, 30 pF
/DS	Output (Input)	Drive: 40 μA/0.4 mA, 30 pF Load: 100 μA/8 mA, 50 pF Open collector, 1 kΩ pull-up
/DSACK0–/DSACK1	In/Out	Load: 100 μA/8 mA, 50 pF Drive: 40 μA/0.2 mA, 30 pF Open collector, 1 kΩ pull-up
ECLK	Output	Drive: 40 μA/0.4 mA, 30 pF
FC0–FC2	Output (Input)	Drive: 20 μA/0.2 mA, 30 pF (Load: 100 μA/8 mA, 50 pF) Open collector, 1 kΩ pull-up

(continued)

■ **Table 15-3** Macintosh IIsi 68030 Direct Slot signals, loading or driving limits (continued)

Signal name	Input/output	Load or drive limits
/HALT	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF Open collector, 1 kΩ pull-up
/IPL0–/IPL2	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF
/IRQ0–/IRQ3	Input	Load: 400 µA/4 mA, 50 pF
/NUBUS	Output	Drive: 40 µA/0.4 mA, 30 pF
PFW	Output	Drive: 40 µA/0.4 mA, 30 pF
/RESET	In/Out	Load: 300 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF Open collector, 1 kΩ pull-up
/RMC	Output	Drive: 40 µA/0.4 mA, 30 pF
R/W	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 400 µA/8 mA, 50 pF) Open collector, 1 kΩ pull-up
SIZ0–SIZ1	Output (Input)	Drive: 40 µA/0.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
/STERM	Output (Input)	Load: 100 µA/8 mA, 50 pF (Load: 40 µA/0.4 mA, 30 pF) Open collector, 1 kΩ pull-up
/TM0A	Input	Load: 400 µA/2 mA, 50 pF
/TM1A	Input	Load: 400 µA/2 mA, 50 pF

Electrical description of the Macintosh IIfx 68030 Direct Slot

Figure 15-2 gives the pinout for the 120-pin expansion connector on the Macintosh IIfx main logic board, as viewed from above.

Table 15-4 lists the pin assignments, gives the signal names, and briefly describes each signal. Table 15-5 provides the load presented or drive available to each pin of an expansion card and indicates whether the signals are inputs or outputs.

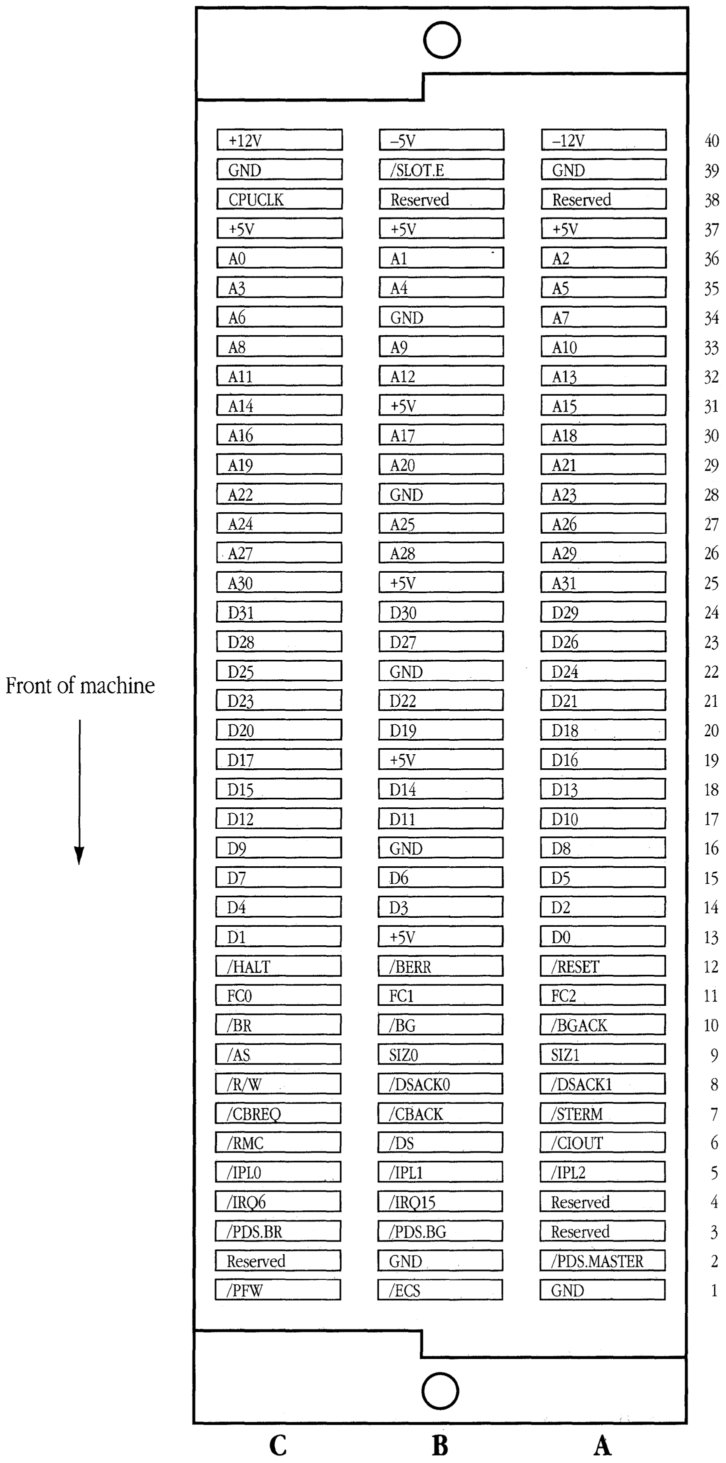
The last column in Table 15-5, labeled "Load or Drive Limits," gives several specifications. An example may be helpful in interpreting this column. The /CBREQ line is shown as presenting a load of 100 µA/5 mA, 50 pF. This is the maximum expected load that an expansion card must drive when sending a /CBREQ signal to the main logic board. The DC load is given in the format *signal high/signal low*. This means that the expansion card must drive a load of up to 100 µA when it drives /CBREQ high (logic 1) and a load of up to 5 mA when it drives /CBREQ low (logic 0). The AC load is given as 50 pF, the maximum capacitance to ground presented by the main logic board to AC signals from an expansion card. The parentheses around "In" and "Load" indicate that the signal is usually driven by the MC68030 processor, but after granting the bus to a DMA requester, the processor tristates the signal and an expansion card may drive it. The notation "Tristate, 1 kΩ pull-up" gives the value of the required pull-up resistor.

In addition, /CBREQ presents a drive of 40 µA/1.2 mA, 50 pF. This is the maximum amount of drive from the main logic board that is available to integrated circuits on the expansion card. The /CBREQ line can drive an expansion card DC load of up to 40 µA in the high state or up to 1.2 mA in the low state. The AC drive is given as 50 pF, the maximum capacitance to ground that an expansion card may present to AC signals from the /CBREQ line.

Next, look at /BERR and you see that only the *signal low* value is given, which means the expansion card must drive a load of up to 48 mA when it drives /BERR low (logic 0). The *signal high* value is not required because the notation "Open collector, 220 Ω pull-up" in the table means that the /BERR line is normally in the open collector state; it is only driven low, and a 220 Ω pull-up resistor on the main logic board returns the line to a logic 1. This is true for all open collector signals in Table 15-5.

Some of the expansion connector signals are specified to drive one 74LS input (a standard 74LS input load is 20 µA high, 0.4 mA low); other signals can drive two 74LS inputs. This differs from the Macintosh SE expansion connector guidelines described in Chapter 13. These strict limitations are imposed to protect the noise and timing margins of the main logic board. All signals needed by an expansion card should be buffered at the expansion connector. The use of newer logic families with very low input loading allows you more margin and flexibility in your expansion card designs.

■ **Figure 15-2** Macintosh IIfx 68030 Direct Slot expansion connector pinout



C	B	A	
+12V	–5V	–12V	40
GND	/SLOT.E	GND	39
CPUCLK	Reserved	Reserved	38
+5V	+5V	+5V	37
A0	A1	A2	36
A3	A4	A5	35
A6	GND	A7	34
A8	A9	A10	33
A11	A12	A13	32
A14	+5V	A15	31
A16	A17	A18	30
A19	A20	A21	29
A22	GND	A23	28
A24	A25	A26	27
A27	A28	A29	26
A30	+5V	A31	25
D31	D30	D29	24
D28	D27	D26	23
D25	GND	D24	22
D23	D22	D21	21
D20	D19	D18	20
D17	+5V	D16	19
D15	D14	D13	18
D12	D11	D10	17
D9	GND	D8	16
D7	D6	D5	15
D4	D3	D2	14
D1	+5V	D0	13
/HALT	/BERR	/RESET	12
FC0	FC1	FC2	11
/BR	/BG	/BGACK	10
/AS	SIZ0	SIZ1	9
/R/W	/DSACK0	/DSACK1	8
/CBREQ	/CBACK	/STERM	7
/RMC	/DS	/CIOUT	6
/IPL0	/IPL1	/IPL2	5
/IRQ6	/IRQ15	Reserved	4
/PDS.BR	/PDS.BG	Reserved	3
Reserved	GND	/PDS.MASTER	2
/PFW	/ECS	GND	1

■ **Table 15-4** Macintosh IIfx 68030 Direct Slot connector signals

Connector		**Signal**	**Signal**
Row	**Pin**	**name**	**description**
A	1	GND	Ground
A	2	/PDS.MASTER	PDS replaces 68030 processor in bus arbitration scheme
A	3	Reserved	For use by Apple
A	4	Reserved	For use by Apple
A	5	/IPL2	Interrupt priority 2
A	6	/CIOUT	Cache inhibit out
A	7	/STERM	Synchronous termination
A	8	/DSACK1	Data acknowledge 1
A	9	SIZ1	Transfer size bit 1
A	10	/BGACK	Bus grant acknowledge
A	11	FC2	Function code 2
A	12	/RESET	System reset
A	13	D0	Data bit 0
A	14	D2	Data bit 2
A	15	D5	Data bit 5
A	16	D8	Data bit 8
A	17	D10	Data bit 10
A	18	D13	Data bit 13
A	19	D16	Data bit 16
A	20	D18	Data bit 18
A	21	D21	Data bit 21
A	22	D24	Data bit 24
A	23	D26	Data bit 26
A	24	D29	Data bit 29
A	25	A31	Address bit 31
A	26	A29	Address bit 29
A	27	A26	Address bit 26
A	28	A23	Address bit 23
A	29	A21	Address bit 21
A	30	A18	Address bit 18
A	31	A15	Address bit 15
A	32	A13	Address bit 13
A	33	A10	Address bit 10
A	34	A7	Address bit 7
A	35	A5	Address bit 5
A	36	A2	Address bit 2
A	37	+5V	5 volts

(continued)

■ **Table 15-4** Macintosh IIfx 68030 Direct Slot connector signals (continued)

Connector Row	 Pin	Signal name	Signal description
A	38	Reserved	For use by Apple
A	39	GND	Ground
A	40	–12V	–12 volts
B	1	/ECS	Early cycle start
B	2	GND	Ground
B	3	/PDS.BG	Bus grant used if /PDS.MASTER is active (low)
B	4	/IRQ15	Interrupt line if pseudoslot design is not used
B	5	/IPL1	Interrupt priority 1
B	6	/DS	Data strobe
B	7	/CBACK	Cache burst acknowledge
B	8	/DSACK0	Data acknowledge 0
B	9	SIZ0	Transfer size bit 0
B	10	/BG	Bus grant to external device
B	11	FC1	Function code 1
B	12	/BERR	Bus error
B	13	+5V	5 volts
B	14	D3	Data bit 3
B	15	D6	Data bit 6
B	16	GND	Ground
B	17	D11	Data bit 11
B	18	D14	Data bit 14
B	19	+5V	5 volts
B	20	D19	Data bit 19
B	21	D22	Data bit 22
B	22	GND	Ground
B	23	D27	Data bit 27
B	24	D30	Data bit 30
B	25	+5V	5 volts
B	26	A28	Address bit 28
B	27	A25	Address bit 25
B	28	GND	Ground
B	29	A20	Address bit 20
B	30	A17	Address bit 17
B	31	+5V	5 volts
B	32	A12	Address bit 12
B	33	A9	Address bit 9
B	34	GND	Ground

(continued)

■ **Table 15-4** Macintosh IIfx 68030 Direct Slot connector signals (continued)

Connector Row	 Pin	Signal name	Signal description
B	35	A4	Address bit 4
B	36	A1	Address bit 1
B	37	+5V	5 volts
B	38	Reserved	For use by Apple
B	39	/SLOT.E	When active (low), the 68030 Direct Slot replaces slot $E in the address map
B	40	–5V	–5 volts
C	1	/PFW	Shutdown bit
C	2	Reserved	For use by Apple
C	3	/PDS.BR	Bus request used if /PDS.MASTER is active (low)
C	4	/IRQ6	PDS interrupt line for pseudoslot $E
C	5	/IPL0	Interrupt priority 0
C	6	/RMC	Read modify cycle
C	7	/CBREQ	Cache burst request
C	8	/R/W	Read/write
C	9	/AS	Address strobe
C	10	/BR	Bus request
C	11	FC0	Function code 0
C	12	/HALT	Halt
C	13	D1	Data bit 1
C	14	D4	Data bit 4
C	15	D7	Data bit 7
C	16	D9	Data bit 9
C	17	D12	Data bit 12
C	18	D15	Data bit 15
C	19	D17	Data bit 17
C	20	D20	Data bit 20
C	21	D23	Data bit 23
C	22	D25	Data bit 25
C	23	D28	Data bit 28
C	24	D31	Data bit 31
C	25	A30	Address bit 30
C	26	A27	Address bit 27
C	27	A24	Address bit 24
C	28	A22	Address bit 22
C	29	A19	Address bit 19
C	30	A16	Address bit 16

(continued)

■ **Table 15-4** Macintosh IIfx 68030 Direct Slot connector signals (continued)

Connector Row	 Pin	Signal name	Signal description
C	31	A14	Address bit 14
C	32	A11	Address bit 11
C	33	A8	Address bit 8
C	34	A6	Address bit 6
C	35	A3	Address bit 3
C	36	A0	Address bit 0
C	37	+5V	5 volts
C	38	CPUCLK	20 MHz CPU clock
C	39	GND	Ground
C	40	+12V	+12 volts

■ **Table 15-5** Macintosh IIfx 68030 Direct Slot signals, loading or driving limits

Signal name	Input/output	Load or drive limits
A0–A31	In/Out	Load: 100 µA/8 mA, 150 pF Drive: 40 µA/1.2 mA, 30 pF
D0–D23	In/Out	Load: 100 µA/8 mA, 130 pF Drive: 40 µA/1.2 mA, 30 pF
D24–D31	In/Out	Load: 100 µA/8 mA, 150 pF Drive: 40 µA/1.2 mA, 30 pF
/AS	(In)/Out	(Load: 100 µA/5 mA, 130 pF) Drive: 80 µA/2.4 mA, 50 pF Tristate, 1 kΩ pull-up
BERR	In/Out	Load: 48 mA, 120 pF Drive: 6 mA, 15 pF (critical) Open collector, 220 Ω pull-up
/BG	Out	Drive: 40 µA/0.6 mA, 30 pF
/BGACK	In/Out	Load: 10 mA, 100 pF Drive: 2 mA, 30 pF Open collector, 470 Ω pull-up
/BR	In	Load: 100 µA/8 mA, 50 pF 3.3 kΩ pull-up

(continued)

■ **Table 15-5** Macintosh IIfx 68030 Direct Slot signals, loading or driving limits (continued)

Signal name	Input/output	Load or drive limits
/CBACK	In/Out	Load: 10 mA, 50 pF Drive: .6 mA, 30 pF Open collector, 470 Ω pull-up
/CBREQ	(In)/Out	(Load: 100 µA/5 mA, 50 pF) Drive: 40 µA/1.2 mA, 50 pF Tristate, 1 kΩ pull-up
/CIOUT	(In)/Out	(Load: 40 µA/1.6 mA, 50 pF) Drive: 20 µA/0.6 mA, 50 pF Tristate, 3.3 kΩ pull-up
CPUCLK	Out	Drive: 80 mA; driven by an emitter follower
/DS	(In)/Out	(Load: 100 µA/5 mA, 100 pF) Drive: 80 µA/2.4 mA, 50 pF Tristate, 1 kΩ pull-up
/DSACK0–/DSACK1	In/Out	Load: 5 mA, 30 pF Drive: 10 mA, 50 pF Open collector, 1 kΩ pull-up
/ECS	(In)/Out	(Load: 100 µA/5 mA, 50 pF) Drive: 20 µA/0.6 mA, 15 pf (critical) Tristate, 1 kΩ pull-up
FC0–FC2	(In)/Out	(Load: 400 µA/4 mA, 80 pF) Drive: 80 µA/2.4 mA, 30 pF Tristate, 3.3 kΩ pull-up
/HALT	In/Out	Load: 48 mA, 100 pF Drive: 6 mA, 15 pF (critical) Open collector, 220 Ω pull-up
/IPL0–/IPL2	Out	Drive: 40 µA/0.4 mA, 30 pF
/PDS.BG	In	Load: 25 µA/250 µA, 50 pF 3.3 kΩ pull-up
/PDS.BR	Out	Drive: 100 µA/8 mA, 50 pF
/PDS.MASTER	In	Load: 25 µA/250 µA, 50 pF
/PFW	Out	Refer to the section "/PFW Interaction With the Power Supply" in Chapter 5 for details

(continued)

■ **Table 15-5** Macintosh IIfx 68030 Direct Slot signals, loading or driving limits (continued)

Signal name	Input/output	Load or drive limits
/RESET	In/Out	Load: 18 µA, 260 pF Drive: 10 mA, 50 pF Open collector, 470 Ω pull-up
/RMC	(In)/Out	(Load: 40 µA/1.6 mA, 50 pF) Drive: 20 µA/0.6 mA, 50 pF Tristate, 3.3 kΩ pull-up
R/W	(In)/Out	(Load: 100 µA/5 mA, 150 pF) Drive: 80 µA/2.4 mA, 30 pF Tristate, 1 kΩ pull-up
SIZ0–SIZ1	(In)/Out	(Load: 100 µA/100 µA, 100 pF) Drive: 40 µA/1.2 mA, 30 pF
/SLOT.E	In	Load: 25 µA/250 µA, 50 pF 3.3 kΩ pull-up
/STERM	In/Out	Load: 16 mA, 100 pF Drive: .6 mA, 15 pF (critical) Open collector, 330 Ω pull-up

Functional description of the MC68030 signals

The Macintosh SE/30, the Macintosh IIsi, and the Macintosh IIfx computers share a common set of address and data signals, and most of the same control, power, and ground signals. Table 15-6 lists those signals that are common to the three machines as well as future 68030-based PDS machines without NuBus. Two of the clock signals (ECLK and C16M) shown in Table 15-6 are not used on the Macintosh IIfx. Features and exceptions that pertain to the Macintosh IIfx only are explained in footnotes.

◆ *Note:* Your Macintosh IIfx expansion card design should include an oscillator for general-purpose timing requirements. Due to loading constraints of the Macintosh IIfx and other high-speed computers, it is impossible to route clock lines over the main logic board, especially to the expansion connector.

In addition to their common signals, each computer includes a group of machine-specific (unique) signals. The Macintosh SE/30 and the Macintosh IIsi share a common set of these machine-specific signals. Pins that are currently defined as reserved may be added to the lists of machine-specific signals on future machines.

■ **Table 15-6** 68030 Direct Slot common signals

Signal name	Description
A0–A31	Address lines.
D0–D31	Data lines.
/AS	A tristate output signal indicating that a valid address is on the bus.
/BERR	A bus error signal indicating that an invalid bus operation is being attempted.†
/BG	An output signal indicating that an external device may become bus master following completion of the current processor bus cycle.
/BGACK	An input signal indicating that an external device has become bus master.
/BR	An input signal indicating that an external device wishes to become bus master.
C16M	A general-purpose 15.6672 MHz clock (not used on the Macintosh IIfx).
/CBACK	An input signal indicating that the accessed device can operate in burst mode.
/CBREQ	A tristate output signal indicating a burst request for the instruction or data cache.
/CIOUT	A tristate output signal that inhibits the operation of an external cache.‡
/DS	Data strobe signal. During a read, /DS indicates that an external device should place valid data on the data bus; during a write, /DS indicates MC68030 has placed valid data on the data bus.
/DSACK0–/DSACK1	Data transfer acknowledge signals that indicate the completion of a data-transfer operation.
ECLK	Main logic board VIA chip clock signal (not used on the Macintosh IIfx).

(continued)

■ **Table 15-6** 68030 Direct Slot common signals (continued)

Signal name	Description
FC0–FC2	Three-bit function code used to identify the address space of current bus cycle.
/HALT	A signal indicating that the processor should suspend bus activity.†
/IPL0–/IPL2	Interrupt priority level lines.§
/PFW	A status signal informing the expansion card that power will be removed. See the section "/PFW Interaction With the Power Supply" in Chapter 5 for details.
/RESET	A bidirectional signal that initiates a system reset.
/RMC	A tristate output signal that identifies the current bus cycle as part of an indivisible read-modify-write operation.
R/W	A tristate output signal that defines the bus transfer as a read or write cycle.
SIZ0–SIZ1	Tristate output signals indicating the number of bytes remaining to be transferred during the current bus cycle.
/STERM	A bus response signal indicating that the addressed port size is 32 bits and that data may be latched on the next falling clock edge for a read cycle.†

† The maximum capacitive load allowed on these signal lines is 15 pF due to the high-speed nature of the Macintosh.

‡ On the Macintosh IIfx, /CIOUT must not be used in conjunction with /CBREQ because the cache should not be inhibited during burst mode cycles.

§ Although these signals are available at the Macintosh IIfx expansion connector, you should not use them in your design. Instead use the PDS interrupt line, /IRQ6, which is tied into the Macintosh IIfx interrupt scheme and can be prioritized, masked, and so on.

Macintosh SE/30 and Macintosh IIsi 68030 Direct Slot machine-specific signals

Table 15-7 lists the 68030 Direct Slot signals that are specific to the Macintosh SE/30 and Macintosh IIsi computers. All of the machine-specific signals listed in Table 15-7 (except the CPUCLK signal) emulate equivalent signals on the NuBus expansion interface. Because of the limited amount of space available in the memory map of the Macintosh SE/30 and Macintosh IIsi computers, you should design your 68030 Direct Slot expansion card to occupy the same 32-bit physical address ranges occupied by NuBus cards in Macintosh computers. This method of emulating NuBus expansion slot address space is called *pseudoslot design.* The pseudoslot interrupt support lines allow the use of the Macintosh Slot Manager driver routines and thus provide an easy software port for NuBus designs. Pseudoslot design is the preferred expansion design strategy for Macintosh computers with both processor-direct and NuBus slots. See the section "Pseudoslot Design Guidelines for Macintosh SE/30 and Macintosh IIsi Expansion Cards," later in this chapter, for more information on pseudoslot design.

Cards that take advantage of these pseudoslot features won't work in a Macintosh NuBus slot because of bus conflicts with physical NuBus. These pseudoslot signal lines will be available on future 68030-based Macintosh PDS computers without physical NuBus. A slightly different pseudoslot signal configuration is used on machines that include both the NuBus and PDS expansion interfaces.

By porting the NuBus design to the 68030 Direct Slot via pseudoslot, you need to supply only one driver for both 68030 Direct Slot and NuBus cards and can design cards that will be usable in future Macintosh computers without NuBus support.

■ **Table 15-7** Macintosh SE/30 and Macintosh IIsi 68030 Direct Slot machine-specific signals

Signal name	Description
/BUSLOCK	NuBus status bit that goes low to signal that an alternate bus master has acquired the bus. Currently not used. This signal is common across Macintosh machines without physical NuBus.
CPUCLK	Provides signal timing and synchronization to ensure compatibility with future versions of the Macintosh. On the Macintosh SE/30, this is a 15.6672 MHz clock, and on the Macintosh IIsi, a 20 MHz clock.
/IRQ1–/IRQ3	General-purpose interrupts that correspond to the three pseudoslot addresses. These signals are common across Macintosh machines without physical NuBus.
/NUBUS	Address decode of the memory range $6000 0000–$FFFF FFFF. Note that this signal is active when the CPU accesses the on-board video display. Expansion cards must further decode the slot address ranges to avoid conflict with the video logic. This signal is common across Macintosh machines without physical NuBus.
/TM0A–/TM1A	Status input signals to VIA2 that are currently not used.

Machine-specific signals for the Macintosh IIfx 68030 Direct Slot

Table 15-8 lists the 68030 Direct Slot signals that are specific to the Macintosh IIfx computer. If you design your PDS expansion card so that the /SLOT.E signal is grounded (low) when it is plugged into the slot, the card automatically looks like a NuBus card occupying slot $E in the Macintosh IIfx address map. This is similar to the pseudoslot design used with Macintosh SE/30 expansion cards, except your card occupies only the 32-bit address space of a NuBus card in slot $E. The Macintosh IIfx uses only one dedicated interrupt line, /IRQ6, to support NuBus pseudoslot $E, while the Macintosh SE/30 uses three interrupt lines to support its NuBus pseudoslot addresses. The Macintosh IIfx also includes another interrupt line, /IRQ15, that you should use if your design does not support the NuBus pseudoslot and you are providing a stand-alone, card-specific software driver.

■ **Table 15-8** Macintosh IIfx machine-specific signals on the 68030 Direct Slot

Signal name	Description
/ECS	A signal from the MC68030 indicating early cycle start.
/IRQ15	An interrupt line that is used if the expansion card does not support the NuBus pseudoslot.
/IRQ6	A dedicated interrupt line, from the processor to the 68030 Direct Slot, that supports NuBus pseudoslot $E. To prevent incompatibility on the Macintosh IIfx, use /IRQ6 instead of input priority level lines /IPL0 through /IPL2.
/PDS.BG	Bus grant signal from PDS expansion card functioning as bus master; it issues this signal to grant the bus to another requester.
PDS.BR	Bus request signal received by PDS expansion card functioning as bus master.
/PDS.MASTER	When this signal is active, the PDS expansion card replaces the MC68030 in the bus arbitration scheme.
/SLOT.E	When active, this signal indicates that the PDS expansion card is replacing NuBus slot $E in the Macintosh IIfx address map.

Electrical design guidelines for Macintosh SE/30 and Macintosh IIsi PDS expansion cards

The following paragraphs provide information that you should become familiar with before starting your expansion card design. Included are a description of how an expansion card gains access to memory and I/O devices, information on pseudoslot design, a description of how the interrupt-handling mechanism works, a summary of design hints, and a discussion of Macintosh SE/30 and Macintosh IIsi expansion card power requirements.

Memory and I/O access from a Macintosh SE/30 expansion card

An expansion card can occupy one of the available unused address locations in the Macintosh SE/30 memory map. See Table 15-9 for a listing of the Macintosh SE/30 memory map's 32-bit physical address space assignments. The Macintosh SE/30 processor can gain access to the expansion card in the same way that it gains access to any of the computer's I/O devices.

Compared with accessing the expansion electronics from the Macintosh SE/30 processor, the task of accessing resources on the main logic board from an expansion card coprocessor is a bit more complex. When an expansion coprocessor needs to access Macintosh SE/30 resources, it requests the bus from the MC68030 using the bus request signal (/BR). The MC68030 grants the bus (/BG) and tristates itself off the bus at the end of that bus cycle. The coprocessor then takes over as bus master (/BGACK). At this point the coprocessor has complete access to all Macintosh SE/30 electronics.

For all devices on the Macintosh SE/30 main logic board, the timing of an access is controlled by the GLU gate array. Once the coprocessor has been given the bus, it asserts a valid address and address strobe to the main logic board. The gate array detects the address and generates the necessary chip selects for the devices. The gate array also generates the /DSACKx signals to inform the coprocessor of cycle completion.

The Macintosh SE/30 design uses the Apple Sound Chip and the SWIM floppy disk controller instead of the discrete sound circuits and the IWM used in the Macintosh SE. Because of this, no extra cycles are required for loading the sound registers or floppy disk speed parameters. Therefore, no special synchronization logic is required in the design of an expansion card.

When accessing RAM and ROM resources on the Macintosh SE/30 logic board, the timing and access requirements are the same as for I/O accesses. This differs from the Macintosh SE because the video RAM is not shared with the system RAM but instead is a separate device residing in a separate address space.

■ **Table 15-9** Macintosh SE/30 32-bit physical address spaces

Address	Description
$0000 0000–$000F FFFF	RAM (minimum configuration)
$0010 0000–$00CF FFFF	RAM (expansion area)
$00D0 0000–$3FFF FFFF	RAM (undefined)
$4000 0000–$4007 FFFF	ROM bank 0 (minimum configuration)
$4008 0000–$4FFF FFFF	ROM (undefined)
$5000 0000–$5000 1FFF	VIA1 (x0200)
$5000 2000–$5000 3FFF	VIA2 (x0200)
$5000 4000–$5000 5FFF	SCC (x0002)
$5000 6000–$5000 7FFF	SCSI (handshake)
$5001 0000–$5001 1FFF	SCSI (x0010)
$5001 2000–$5001 3FFF	SCSI (pseudo-DMA)
$5001 4000–$5001 5FFF	Sound
$5001 6000–$5001 7FFF	SWIM
$5001 8000–$57FF FFFF	Undefined
$5800 0000–$5FFF FFFF	68030 Direct Slot expansion (if pseudoslot is not used)
$6000 0000–$EFFF FFFF	NuBus super slot space (256 MB per slot)
$F000 0000–$F8FF FFFF	Expansion (undefined)
$F900 0000–$FBFF FFFF	Expansion pseudoslot space (emulate NuBus)
$FC00 0000–$FDFF FFFF	Expansion (undefined)
$FE00 0000–$FE00 FFFF	Video RAM space
$FEFF 0000–$FEFF FFFF	Video ROM space
$FF00 0000–$FFFF FFFF	Expansion (undefined)

◆ *Note:* When the overlay signal is true at boot time, the RAM is not accessible by the processor and the ROM resides at address 0 and its normal location. When overlay is false, the mapping in Table 15-9 is valid.

Memory and I/O access from Macintosh IIsi expansion cards

An expansion card in the Macintosh IIsi can occupy 32-bit addresses from $F900 0000 through $FBFF FFFF. This is equivalent to geographic NuBus locations $9, $A, and $B. See Table 15-10 for a listing of the Macintosh IIsi memory map's physical address space assignments. If you are designing a new processor-direct expansion card for the Macintosh IIsi, you should use the pseudoslot design method to emulate this NuBus address space. The pseudoslot design method is discussed in a later section of this chapter, "Pseudoslot Design Guidelines for Macintosh SE/30 and Macintosh IIsi Expansion Cards."

Accessing resources on the main logic board from a Macintosh IIsi expansion card is identical to accessing resources on the Macintosh SE/30. When an expansion coprocessor needs to access Macintosh IIsi resources, it requests the bus from the MC68030 using the bus request signal (/BR). The MC68030 grants the bus (/BG) and tristates itself off the bus at the end of that bus cycle. The coprocessor then takes over as bus master (/BGACK). At this point, the coprocessor has complete access to all Macintosh IIsi electronics.

Like the Macintosh SE/30, the Macintosh IIsi design uses the Apple Sound Chip and the SWIM floppy disk controller instead of the discrete sound circuits and the IWM used in the Macintosh SE. Because of this, no extra cycles are required for loading the sound registers or floppy disk speed parameters. Therefore, no special synchronization logic is required in the design of an expansion card.

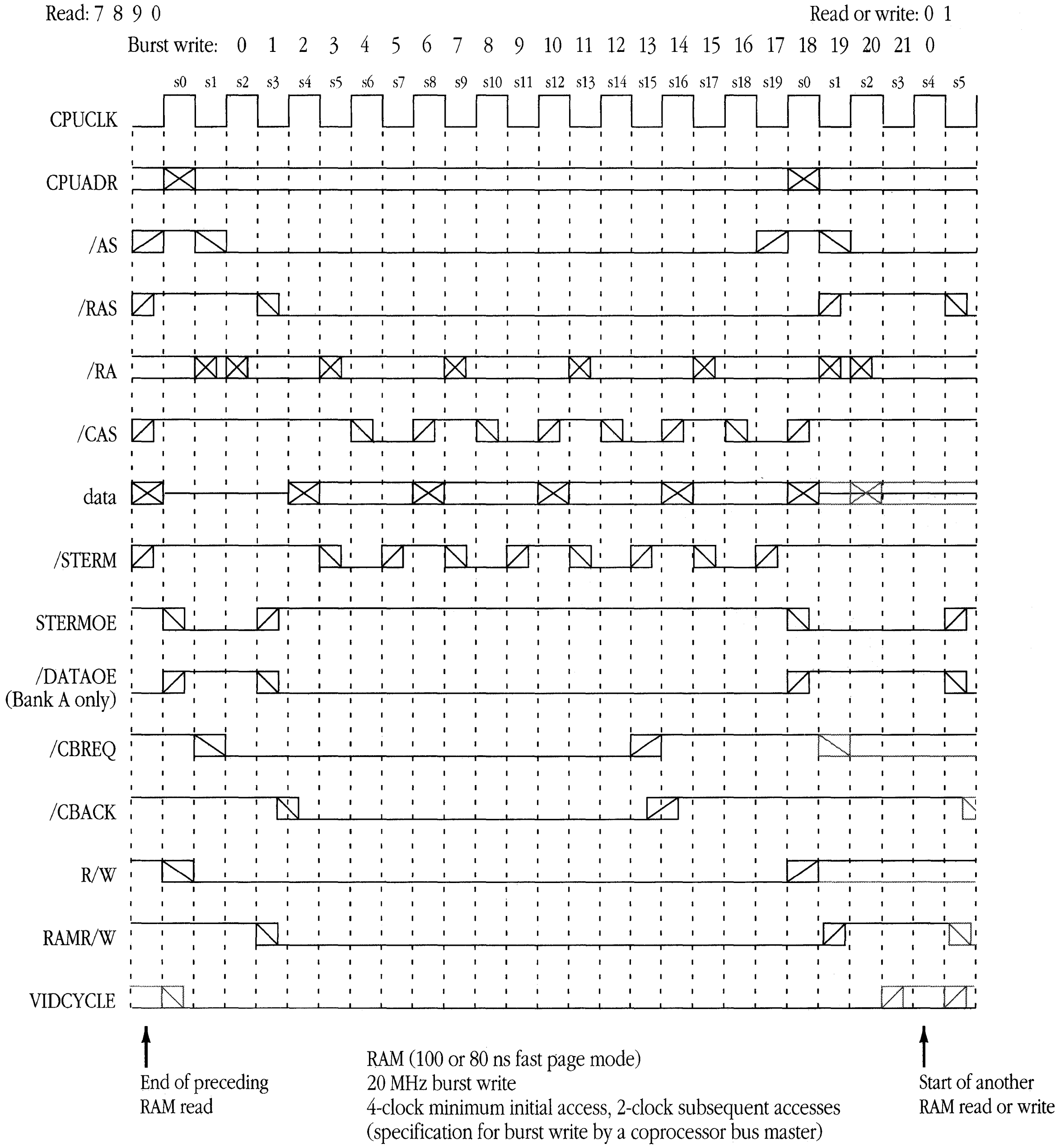
■ **Table 15-10** Macintosh IIsi 32-bit physical address spaces

Address	Description
$0000 0000–$03FF FFFF	RAM Bank A (minimum configuration)
$0400 0000–$07FF FFFF	RAM Bank B
$0800 0000–$3FFF FFFF	RAM (expansion area)
$4000 0000–$4FFF FFFF	ROM
$5000 0000–$5000 1FFF	VIA1
$5000 2000–$5000 3FFF	Reserved
$5000 4000–$5000 5FFF	SCC
$5000 6000–$5000 7FFF	SCSI (pseudo-DMA with DRO)
$5000 8000–$5000 FFFF	Reserved
$5001 0000–$5001 1FFF	SCSI (normal mode)
$5001 2000–$5001 3FFF	SCSI (pseudo-DMA with no DRO)
$5001 4000–$5001 5FFF	SOUND
$5001 6000–$5001 7FFF	SWIM
$5001 8000–$5002 3FFF	Reserved
$5002 4000–$5002 5FFF	VDAC
$5002 6000–$5002 7FFF	RBV
$5002 8000–$5FFF FFFF	Reserved
$6000 0000–$EFFF FFFF	NuBus super slots
$F000 0000–$F0FF FFFF	Reserved
$F100 0000–$FFFF FFFF	NuBus slots

RAM access from a PDS expansion card in the Macintosh IIsi

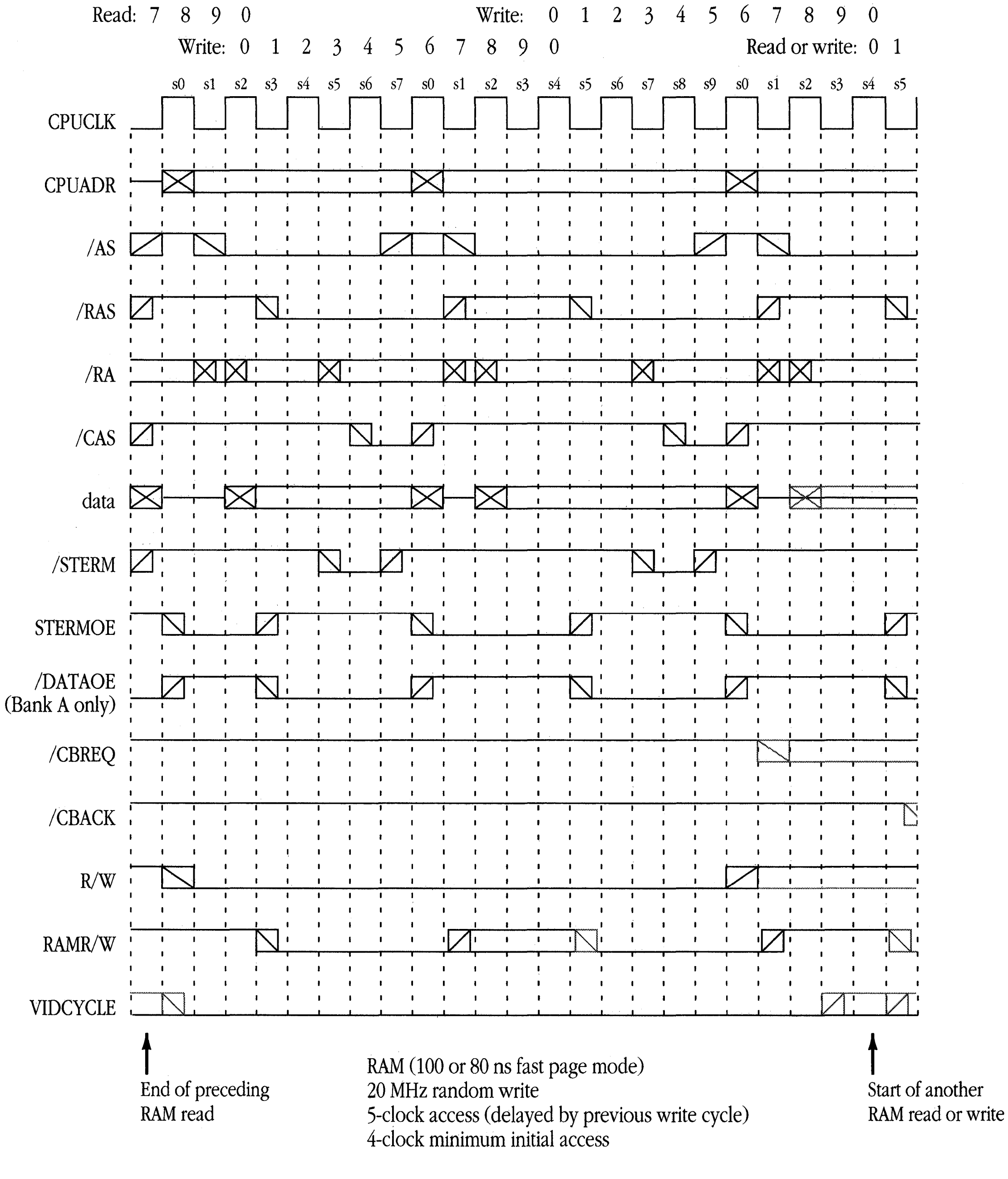
The memory cycle for a Macintosh IIsi processor-direct expansion card operating as bus master is substantially different from that of a Macintosh SE/30 computer. It has been changed to support burst transfers using the /STERM signal generated by the MDU rather than the /DSACK signal generated by the general logic unit chip. If bus master cards look only for /DSACK, they will not work. Figures 15-3 through 15-6 show the timing for both burst and random writes to RAM and reads from RAM.

■ **Figure 15-3** Macintosh IIsi RAM burst-write timing

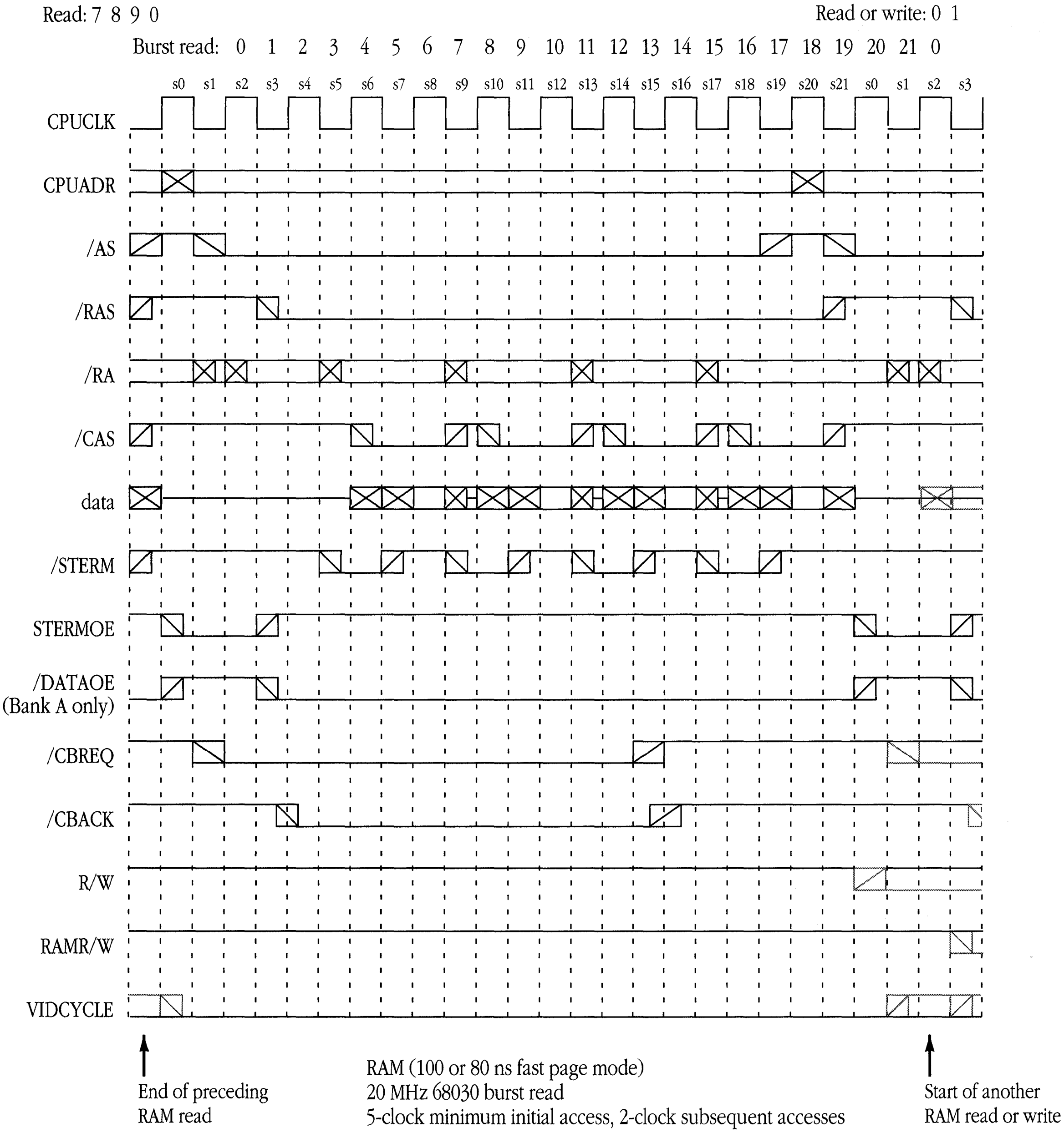

Read: 7 8 9 0
Read or write: 0 1
Burst write: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 0
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s0 s1 s2 s3 s4 s5
CPUCLK
CPUADR
/AS
/RAS
/RA
/CAS
data
/STERM
STERMOE
/DATAOE
(Bank A only)
/CBREQ
/CBACK
R/W
RAMR/W
VIDCYCLE
End of preceding RAM read
Start of another RAM read or write

RAM (100 or 80 ns fast page mode)
20 MHz burst write
4-clock minimum initial access, 2-clock subsequent accesses
(specification for burst write by a coprocessor bus master)

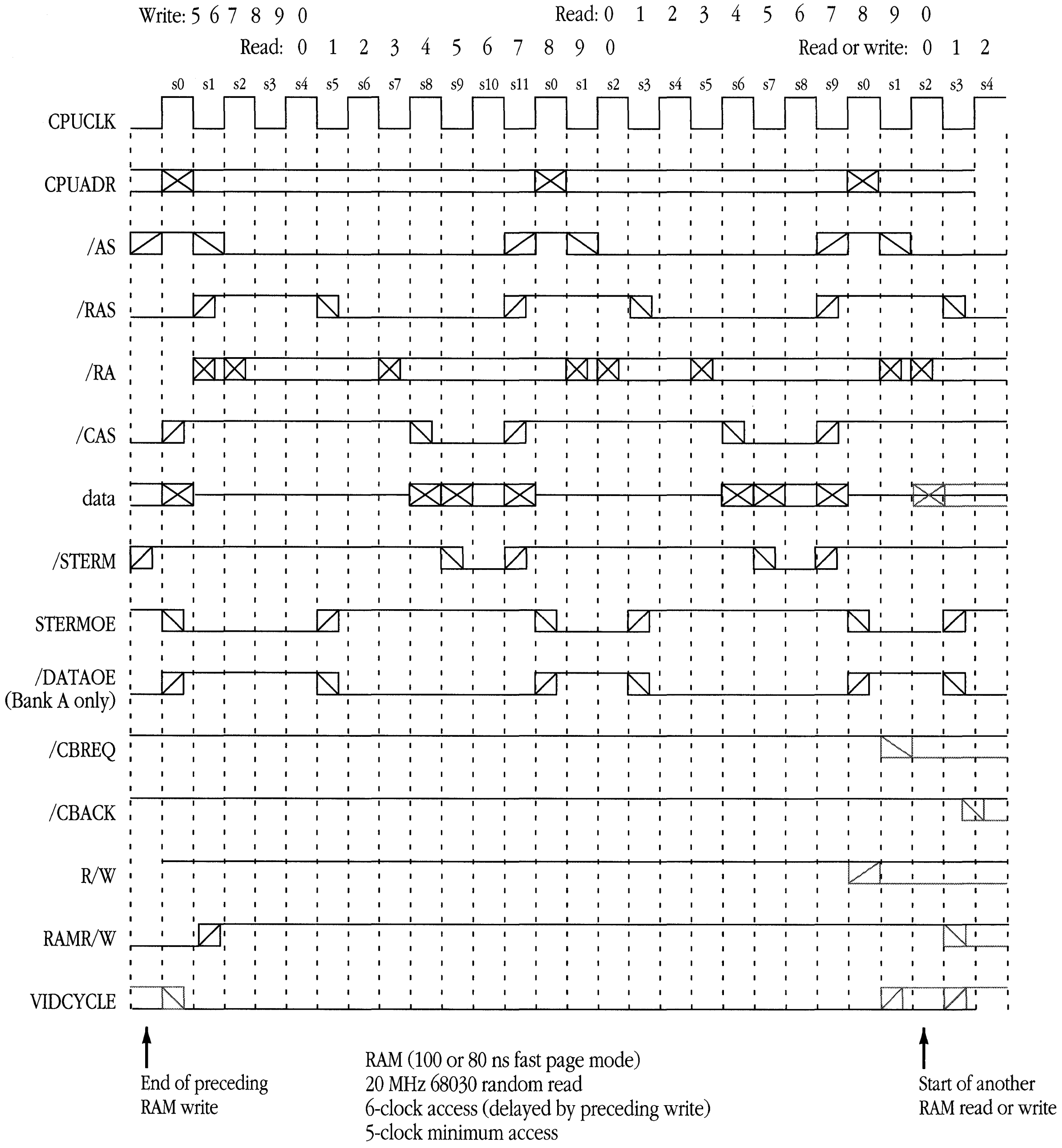
■ **Figure 15-4** Macintosh IIsi RAM random-write timing


Read: 7 8 9 0
Write: 0 1 2 3 4 5 6 7 8 9 0
Write: 0 1 2 3 4 5 6 7 8 9 0
Read or write: 0 1
s0 s1 s2 s3 s4 s5 s6 s7 s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s0 s1 s2 s3 s4 s5
CPUCLK
CPUADR
/AS
/RAS
/RA
/CAS
data
/STERM
STERMOE
/DATAOE (Bank A only)
/CBREQ
/CBACK
R/W
RAMR/W
VIDCYCLE
End of preceding RAM read
RAM (100 or 80 ns fast page mode)
20 MHz random write
5-clock access (delayed by previous write cycle)
4-clock minimum initial access
Start of another RAM read or write

■ **Figure 15-5** Macintosh IIsi RAM burst-read timing


Read: 7 8 9 0
Read or write: 0 1
Burst read: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 0
s0 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s0 s1 s2 s3
CPUCLK
CPUADR
/AS
/RAS
/RA
/CAS
data
/STERM
STERMOE
/DATAOE
(Bank A only)
/CBREQ
/CBACK
R/W
RAMR/W
VIDCYCLE
End of preceding
RAM read
RAM (100 or 80 ns fast page mode)
20 MHz 68030 burst read
5-clock minimum initial access, 2-clock subsequent accesses
Start of another
RAM read or write

■ **Figure 15-6** Macintosh IIsi RAM random-read timing



Pseudoslot design guidelines for Macintosh SE/30 and Macintosh IIsi expansion cards

If you are familiar with designing devices for the MC68000 family of microprocessors, you should find it relatively easy to use the pseudoslot method to design an expansion card for the Macintosh SE/30 or the Macintosh IIsi. The only added constraints are the need for a declaration ROM and adherence to some address decoding rules.

Many of the address locations correspond to address ranges used by NuBus expansion cards resident in Macintosh computers that offer NuBus. The advantage of designing an expansion card to occupy one of these unused addresses is that existing ROM firmware with the ability to manage the NuBus slots is also present in your computer's system ROM. Therefore, if an expansion card is designed along the lines of a NuBus card (for example, with a declaration ROM and interrupt capability), the existing Slot Manager ROM firmware controls this card as if it were a NuBus card but the electrical interface is via the MC68030 bus. As a side benefit of this design, one software driver works on machines with two different methods of expansion. To find out more about the Slot Manager, refer to Chapter 8, "NuBus Card Firmware," in this book and the Slot Manager information in *Inside Macintosh.*

If you do not use pseudoslot design, the area from $5800 0000 through $5FFF FFFF in the Macintosh SE/30 and the Macintosh IIsi is reserved as the preferred location for 68030 Direct Slot expansion. If you use this area, your expansion card will work in future Macintosh PDS machines that do not have NuBus. Note that to access this address space, the Macintosh must be in 32-bit mode. It is the responsibility of the card driver to switch between the 24-bit and 32-bit modes using the trap macros `_SwapMMUMode` and `_GetMMUMode`, as defined in the operating-system utilities information in *Inside Macintosh.* You will not be able to use the Slot Manager and must provide card-specific drivers to use this memory area for card expansion. The conversion addresses for the 24-to-32-bit logical address translation are listed in Table 15-11.

■ **Table 15-11** 24-to-32-bit logical address translation map

24-bit address range	32-bit address range
$00 0000–$7F FFFF	$0000 0000–$007F FFFF
$80 0000–$8F FFFF	$4000 0000–$400F FFFF
$90 0000–$9F FFFF	$F900 0000–$F90F FFFF
$A0 0000–$AF FFFF	$FA00 0000–$FA0F FFFF
$B0 0000–$BF FFFF	$FB00 0000–$FB0F FFFF
$C0 0000–$CF FFFF	$FC00 0000–$FC0F FFFF
$D0 0000–$DF FFFF	$FD00 0000–$FD0F FFFF
$E0 0000–$EF FFFF	$FE00 0000–$FE0F FFFF
$F0 0000–$FF FFFF	$5000 0000–$500F FFFF

In many ways, designing an expansion card for the 68030 Direct Slot is simpler than designing one for the NuBus. The 32-bit data bus of the Macintosh SE/30 and Macintosh IIsi supports dynamic bus sizing, so I/O ports of 8, 16, or 32 bits can be designed. Proper control of the /DSACKx signals informs the processor of the bus width, so additional memory cycles can be executed to complete the transfer if necessary. There is no byte swapping between the MC68030 and the expansion connector, and separate address and data buses eliminate the need for address latches.

The Macintosh SE/30 and the Macintosh IIsi expansion slots provide three general-purpose interrupt inputs to the main logic. These interrupts correspond to the first three NuBus slots of a Macintosh II. To have your expansion card reside in address spaces that emulate the NuBus slots, design your card's hardware to use only the physical 32-bit space address ranges shown in Table 15-12 and the software to operate only in the 32-bit mode.

△ **Important** If you are designing a video card, remember that the ROM in the Macintosh SE/30 includes only the 24-bit version of Color QuickDraw. To allow your video card to operate in 32-bit mode, you must bundle the RAM-based version of 32-bit Color QuickDraw with your card. △

To ensure compatibility with future hardware and software, you should decode all the address bits to minimize the chance for address conflicts.

The declaration ROM must reside at the upper address limit of the 16 MB address space in order for the Slot Manager code to recognize the card. Chapter 8, "NuBus Card Firmware," provides information to help you develop the necessary card firmware.

You are not required to follow the pseudoslot method for designing an I/O expansion card. This method is provided as a means to simplify the design task and to minimize the need for revisions of support software.

■ **Table 15-12** Pseudoslot address ranges for Macintosh SE/30 and Macintosh IIsi expansion cards

Interrupt	32-bit address space
1	$F900 0000–$F9FF FFFF
2†	$FA00 0000–$FAFF FFFF
3†	$FB00 0000–$FBFF FFFF

† The Macintosh IIsi can only map its PDS expansion card to the corresponding NuBus slot $9. The other two slots and interrupts cannot be used for the Macintosh IIsi.

Interrupt handling for the Macintosh SE/30 and Macintosh IIsi 68030 Direct Slot

The interrupt-handling mechanism for the 68030 Direct Slot on the Macintosh SE/30 and Macintosh IIsi is similar to the mechanism used in Macintosh computers with NuBus. Here is how the mechanism works. First, the three general-purpose interrupt signals on the 68030 Direct Slot and the on-board video interrupt signal are routed through an OR gate to generate a signal called /SLOTIRQ. This signal is connected to the CA1 input of VIA2, the second VIA chip on the logic board. This VIA generates a level 2 interrupt to the MC68030. This VIA can also generate an interrupt in response to SCSI requests, sound chip requests, or VIA timer requests.

All interrupts to the MC68030 are autovectored using addresses that contain the interrupt vectors. When the MC68030 is executing a level *x* interrupt, it first sets the interrupt mask to level *x,* so further interrupts at level *x* and below will be ignored. Once the interrupt handler is executed and an RTE instruction is processed, the interrupt mask is restored to the value it had before the interrupt.

The first-level interrupt dispatcher determines which hardware device—SCSI, sound chip, real-time clock, or expansion slot—is requesting the interrupt and dispatches code to the appropriate interrupt handler. If the interrupt generated by the VIA is a slot interrupt, the software polls the second VIA, bits PA0 through PA5, to determine which slot generated the interrupt. PA0 is equal to IRQ1, PA1 is equal to IRQ2, and PA2 is equal to IRQ3. PA5 is equal to the video interrupt.

Once the software determines which pseudoslot generated the interrupt, the Slot Manager software executes the interrupt handler for that slot device. The handler for that device was installed at boot time, when the initialization software polled the possible slots and identified the existence of a card in the slot by its ROM signature.

Since all interrupts to the MC68030 are autovectored, care must be exercised in the detection of the processor's interrupt acknowledge. The MC68030 starts an interrupt acknowledge cycle before it checks the level of the AVEC (autovector) pin. Once the processor determines the AVEC pin is signaling an autovector, it aborts the bus cycle without the assertion of /DSACK or /STERM. Hardware designers must be aware of this abort cycle.

There is a delay between the assertion of a slot interrupt and the actual execution of the interrupt handler. During this time, the software polls the actual slot /IRQ signal. The recommended design practice is to latch the slot /IRQ signal so that once it is asserted, the interrupt handler software for the card has the responsibility of clearing the interrupt. This ensures that the slot /IRQ signal is asserted when polled and the Slot Manager is dispatched correctly.

Design hints for Macintosh SE/30 and Macintosh IIsi expansion cards

When designing a card for the Macintosh SE/30 or Macintosh IIsi, you must generate timing to match the requirements of the MC68030 microprocessor. For further information on the timing requirements of the microprocessor, refer to the Motorola *MC68030 Enhanced 32-Bit Microprocessor User's Manual.*

There is an overriding watchdog timer on the Macintosh SE/30 and the Macintosh IIsi main logic boards that generates a /BERR signal any time the address strobe is asserted for longer than 44 µs. You must guarantee that your design generates a /DSACKx, /BERR, or other termination signal within this period.

Notice that there are two clock signals present on the expansion connector. The CPUCLK signal should be used for signal timing and synchronization to ensure compatibility with future versions of the Macintosh that may use a faster CPU clock. The C16M signal is a general-purpose 15.6672 MHz clock that will be present in future machines. In the Macintosh SE/30, these two clocks have the same frequency and phase relationship; however, the Macintosh IIsi uses a 20 MHz CPUCLK signal. In the future, the CPUCLK rate may change further. Because the clock rate can vary, try to design expansion cards to be asynchronous with respect to CPUCLK, or perhaps use a fixed 15.6672 MHz clock to be compatible across different machines.

The data strobe signal is provided for developers of expansion cards that function as DMA masters. The data strobe must be asserted when the DMA master is accessing devices on the Macintosh SE/30 and the Macintosh IIsi main logic boards. The timing of the data strobe should match the MC68030 data strobe signal.

Notice that the /NUBUS signal (Table 15-7) is an address decode of the memory range $6000 0000 through $FFFF FFFF. The /AS (address strobe) signal qualifies the assertion of the /NUBUS signal. The /NUBUS signal is asserted a maximum of 26 ns after the /AS signal is asserted, and is removed a maximum of 22 ns after the /AS signal is removed.

Remember that /NUBUS is valid when the processor is accessing the on-board video logic; therefore, to avoid possible data bus conflicts, you must decode one of the pseudoslot address ranges when using the /NUBUS signal as a qualifier.

The pseudoslot interrupt signals (/IRQ1 through /IRQ3) are active-low TTL-compatible inputs to the main logic board. You do not have to use an open-collector style driver, but if you do, you should provide a pull-up resistor on the expansion card.

If you are designing a bus master card and are accessing on-board devices such as RAM, you must ensure that a DMA cycle is completed when the normal MC68030 processor cycle is completed.

Because of the dynamic bus sizing feature of the MC68030, you can convert existing Macintosh SE expansion cards to fit the Macintosh SE/30 32-bit slot with relative ease. The mechanical changes are probably more extensive than the electrical changes. The Macintosh SE/30 expansion card is mounted vertically rather than horizontally. You can design an adapter card to convert the 120-pin expansion slot to a Macintosh SE–compatible 96-pin expansion slot. The Macintosh SE card could then be piggyback connected to the adapter card. The logic to convert most simple Macintosh SE cards to the 32-bit Macintosh SE/30 design is relatively straightforward and could prove a quick and easy way to convert existing designs to the Macintosh SE/30.

Power consumption guidelines for Macintosh SE/30 and Macintosh IIsi PDS expansion cards

The Macintosh SE/30 and Macintosh IIsi use the same power supply as the Macintosh SE. Therefore, the same power consumption guidelines should be followed. The Macintosh SE power budget is described in Chapter 13 in the section "Power Consumption Guidelines for Macintosh SE PDS Expansion Cards." The Macintosh SE/30 and the Macintosh IIsi main logic boards consume more power than the Macintosh SE main logic board, but if you adhere to the following guidelines there is still enough power supply margin to ensure reliability. Table 15-13 shows the allotted current for an expansion card.

◆ *Note:* For thermal considerations, the total power of the expansion card should not exceed 7 W.

■ **Table 15-13** Power budget for a Macintosh SE/30 and Macintosh IIsi expansion card

Voltage, V	Current load, A
+5	1.5
–5	0.1
+12	0.15
–12	0.1

△ **Important** Seriously consider whether routing power outside the case is necessary. If power is required outside the case, use a fast-acting fuse to protect against an overcurrent load. A fast-acting fuse (1 A) retains the product safety compliance designed into the Macintosh SE/30 and the Macintosh IIsi. △

Macintosh IIfx expansion card design

This section provides technical information that you need to design a PDS expansion card for the Macintosh IIfx computer. Topics covered include pseudoslot design, termination of memory cycles, the interrupt-handling mechanism, the bus priority scheme, the effect of clock speeds on expansion card design, the use of cache memory, and power consumption guidelines.

Pseudoslot design guidelines for Macintosh IIfx PDS expansion cards

It is relatively easy to use the pseudoslot design method to design an expansion card for the Macintosh IIfx 68030 Direct Slot. In addition to making sure that the /SLOT.E signal is held low (grounded), the only constraints are the need for a declaration ROM and adherence to address decoding rules. If you design your card along the lines of a NuBus card (for example, so that it occupies slot $E and has a declaration ROM and interrupt capability), the existing Slot Manager firmware in the system ROM controls the card as if it were a NuBus card, but the electrical interface is via the 68030 Direct Slot. This means that you do not have to develop another software driver; the driver for the NuBus expansion interface will also work with your PDS expansion card.

If you do not use pseudoslot design, your expansion card can occupy either the slow slot space area ($6000 0000 through $6FFF FFFF) or the fast slot space area ($7000 0000 through $7FFF FFFF) in the address map. However, your card cannot communicate with the Slot Manager. You must provide a card-specific driver, and you should use /IRQ15 as your interrupt line.

Memory cycle termination in the Macintosh IIfx

The 32-bit data bus of the Macintosh IIfx supports dynamic bus sizing, so I/O ports of 8, 16, or 32 bits can be designed. Proper control of the /DSACKx signals informs the processor of the bus width, so additional memory cycles can be executed to complete the transfer if necessary. There is no byte swapping between the MC68030 and the expansion connector, and separate address and data buses eliminate the need for address latches. Outgoing memory cycles from the Macintosh IIfx processor support dynamic bus sizing and are terminated by the /DSACK0, /DSACK1, and /STERM signals on the PDS connector. The reverse, however, is not true. Cycles incoming to the Macintosh IIfx memory are 32-bit synchronous and are terminated only by /STERM. Cycles from the PDS expansion card to I/O devices are terminated by /DSACK1 and /DSACK0, except to NuBus, where all reads and aligned longword writes are terminated by /STERM.

An overriding timer on the main logic board generates a /BERR signal anytime the address strobe (/AS) is asserted for longer than 16 µs. Your expansion card design must include a provision for generating /DSACK, /BERR, or other terminating signals within this period.

Interrupt handling for the Macintosh IIfx 68030 Direct Slot

The interrupt-handling mechanism for the Macintosh IIfx 68030 Direct Slot differs from that of previous Macintosh computers with processor-direct slots. The major difference is that the VIA2 in the earlier machines has been eliminated from the high-end Macintosh IIfx computer architecture. It is replaced by the Operating System Support (OSS) chip, an Apple custom IC with two dedicated interrupt lines, /IRQ6 and /IRQ15, to the 68030 Direct Slot. Your expansion card should no longer use interrupt priority lines /IPL2 through /IPL0 or it will be incompatible with the Macintosh IIfx firmware.

The levels of the /IRQ6 and /IRQ15 interrupt lines are fully programmable to provide maximum design flexibility. If you use the pseudoslot method to design your card and it is properly configured so that it can be recognized by the Slot Manager, then the Slot Manager fields all interrupts on the /IRQ6 line as slot $E interrupts. If you do not use the pseudoslot design method, all interrupts on the /IRQ15 line are fielded as nonslot $E interrupts.

Bus master priority scheme for the Macintosh IIfx

It is possible to have multiple bus masters on the Macintosh IIfx processor bus. The possible bus masters and their position in the priority scheme are shown in Table 15-14. Note that the NuBus and SCSI interfaces allow DMA access to the 68030 Direct Slot.

Because the 68030 processor is the lowest-priority bus master, note that as each expansion slot (NuBus and PDS slots) in the Macintosh IIfx is filled, the 68030 processor performance is degraded.

■ **Table 15-14** Macintosh IIfx bus master priority scheme

Priority level	Bus master
First (highest)	68030 Direct Slot
Second	NuBus
Third	SCSI
Fourth (lowest)	MC68030 processor

Effect of Macintosh IIfx clock speeds on PDS expansion card design

The Macintosh IIfx computer consists of two subsystems, the memory (fast) subsystem and the I/O (slow) subsystem. These subsystems are separated by fast/slow buffers. See the block diagram in Figure 1-4.

Timing is provided by an 80 MHz oscillator whose output is divided by 2, resulting in a 40 MHz CPU clock for the memory subsystem. The output of the 80 MHz oscillator is divided by 4 to provide a 20 MHz clock for the I/O subsystem.

Although the 68030 Direct Slot is in the I/O subsystem, it is still classified as a processor-direct slot because when an expansion card addresses the memory subsystem, that subsystem responds in the same amount of time as if the MC68030 processor had addressed it. This same access speed is always maintained because the memory controller speed is constant. Even though the clock supplied to the 68030 Direct Slot is only 20 MHz, a PDS expansion card benefits from the high-speed design of the Macintosh IIfx.

The CPUCLK signal is provided to a PDS expansion card to allow the card to synchronize to the computer. The timing interface looks exactly like the MC68030 processor running at 20 MHz. Since the processor and memory subsystem CPU clock are actually running at 40 MHz, the processor slows down and synchronizes to the 20 MHz clock provided to the 68030 Direct Slot whenever an attempt is made to gain access to the expansion card. This speed shift is transparent to the expansion card, but it can be controlled by the address space that you choose when designing your card. The processor shifts speed if your design uses pseudoslot address space $Exxx xxxx or $FExx xxxx. It also shifts speed if you do not use pseudoslot address space but write your own driver and use slow address space, $6xxx xxxx.

As an option, you may choose to write your own driver and use fast space, $7xxx xxxx. In this case, the CPUCLK signal runs at 20 MHz, but the processor continues to run at 40 MHz and does not slow down to synchronize with the expansion card's 20 MHz clock. You can gain access to the expansion card faster, but design of the card will be more difficult since the processor runs at 40 MHz and you have only a 20 MHz clock to work with. In this configuration the processor-direct slot is phase synchronous with frequencies of 80 MHz and 40 MHz.

As another option, you could include an oscillator on your card that runs at the desired speed, and then double-rank synchronize all signals running between the processor and the 68030 Direct Slot. You can implement double-rank synchronization by running asynchronous signals through two ranks of D type flip-flops that are being clocked at the same frequency that the incoming signals are being synchronized to. The disadvantage of this option is the loss of time created by the double-rank synchronization process.

Yet another design option you may want to consider is phase locking to the 40 MHz clock of the memory subsystem.

Using the Macintosh IIfx cache memory

The addition of the high-speed cache memory makes possible the high-performance characteristics of the Macintosh IIfx computer. The cache is designed so that it is always logically related to the main memory. The cache is fairly large, consisting of 32 KB in a direct-mapped arrangement with 2000 lines of four longwords each. Writes are usually no-wait state cycles and always update the cache at the same time main memory is being updated. Only burst reads are cached.

The memory subsystem in the Macintosh IIfx supports the 68030 cache burst protocol. That is, a PDS expansion card in the 68030 Direct Slot can use /CBREQ to request the main memory to supply four longwords in succession. See the Motorola *MC68030 Enhanced 32-Bit Microprocessor User's Manual* for detailed information and timing. The cache cannot be inhibited during burst cycles, because /CBREQ and /CIOUT are mutually exclusive.

In some systems thrashing can occur as the cache switches back and forth between the 68030 Direct Slot and the processor data, but this is not a problem in the Macintosh IIfx, because of the large size of the cache.

The greatest data-transfer speeds are obtained if you write all code in aligned longwords. The processor still supports bytes, words, misaligned words, and longwords, but the processor must execute multiple cycles to gain access to code written in this manner. Also, you should keep back-to-back writes on the same memory page. Since the fast-memory controller in the Macintosh IIfx has a same-page detector, it does page mode writes if it detects back-to-back writes on the same page, resulting in faster write operations.

Additional design hints

If you are designing a PDS card to operate as bus master and are accessing on-board devices such as RAM, you must ensure that a DMA cycle is completed when the normal MC68030 processor cycle is completed.

The data strobe signal is provided for expansion cards that function as DMA masters. The /DS signal must be asserted when the DMA master is addressing devices on the main logic board. The timing of the data strobe should match the MC68030 data strobe signal.

▲ **Warning** On the Macintosh IIfx, it is not possible for a PDS bus master to read data from a NuBus expansion card. ▲

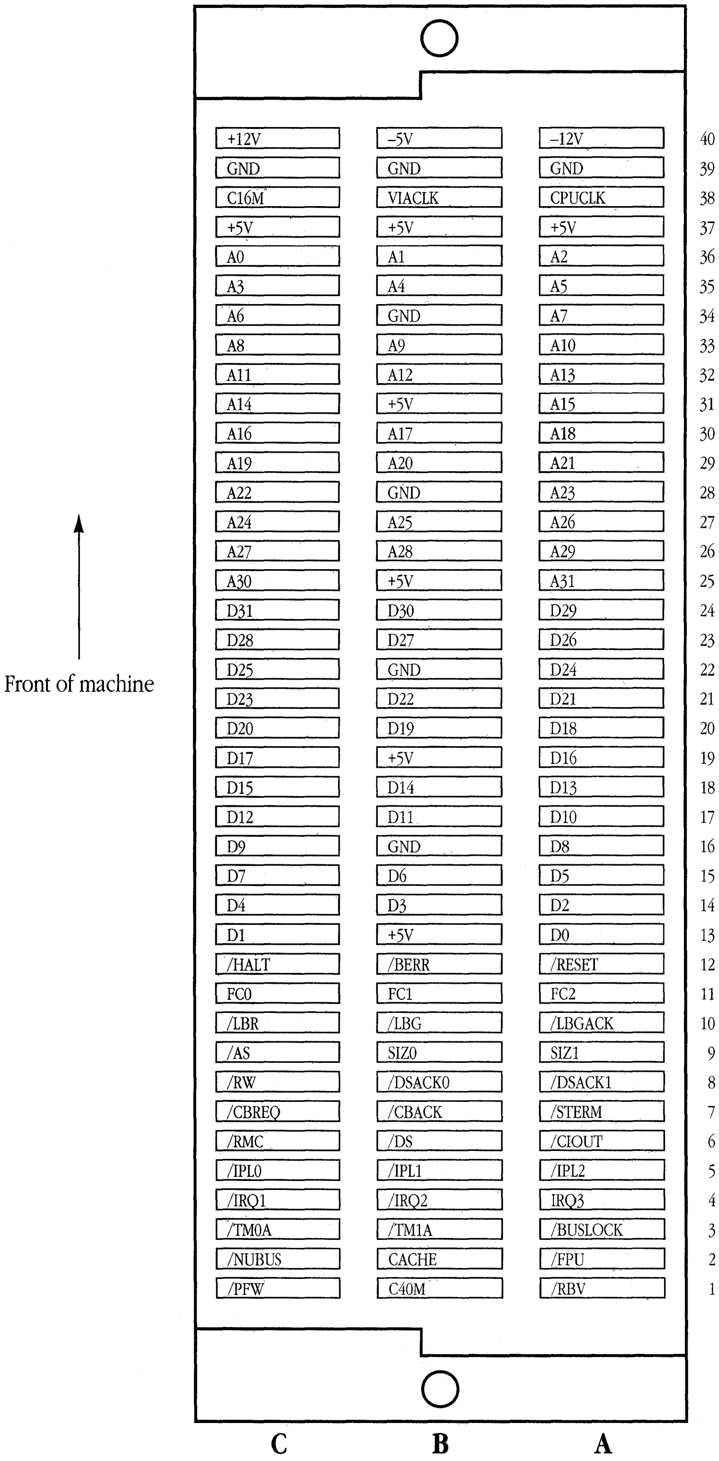
Power consumption guidelines for Macintosh IIfx PDS expansion cards

The power budget for a PDS expansion card in the Macintosh IIfx is identical to the power budget for the NuBus card that it replaces. Refer to the section "NuBus Power Budget" in Chapter 5 for details.

Macintosh IIsi PDS adapter card

To install a processor-direct expansion card on the Macintosh IIsi, you must first install an adapter card. An MC68030 adapter kit is available from authorized Apple dealers, though you may decide to develop your own adapter card. If you decide to develop your own custom adapter card for the Macintosh IIsi, refer to Figure 15-7 to see the expansion connector on the main logic board in which the adapter fits. For a physical description of the Macintosh IIsi adapter card, refer to Chapter 17.

Figure 15-7 Macintosh IIsi main logic board expansion connector



C	B	A	
+12V	–5V	–12V	40
GND	GND	GND	39
C16M	VIACLK	CPUCLK	38
+5V	+5V	+5V	37
A0	A1	A2	36
A3	A4	A5	35
A6	GND	A7	34
A8	A9	A10	33
A11	A12	A13	32
A14	+5V	A15	31
A16	A17	A18	30
A19	A20	A21	29
A22	GND	A23	28
A24	A25	A26	27
A27	A28	A29	26
A30	+5V	A31	25
D31	D30	D29	24
D28	D27	D26	23
D25	GND	D24	22
D23	D22	D21	21
D20	D19	D18	20
D17	+5V	D16	19
D15	D14	D13	18
D12	D11	D10	17
D9	GND	D8	16
D7	D6	D5	15
D4	D3	D2	14
D1	+5V	D0	13
/HALT	/BERR	/RESET	12
FC0	FC1	FC2	11
/LBR	/LBG	/LBGACK	10
/AS	SIZ0	SIZ1	9
/RW	/DSACK0	/DSACK1	8
/CBREQ	/CBACK	/STERM	7
/RMC	/DS	/CIOUT	6
/IPL0	/IPL1	/IPL2	5
/IRQ1	/IRQ2	IRQ3	4
/TM0A	/TM1A	/BUSLOCK	3
/NUBUS	CACHE	/FPU	2
/PFW	C40M	/RBV	1

Most of the signals present in the expansion connector on the main logic board have been connected to the PDS expansion slot on the Apple adapter card. The signals listed in Table 15-15, however, are not present on the Macintosh IIsi MC68030 PDS adapter card. These signals should only be used for the development of the adapter card.

■ **Table 15-15** Macintosh IIsi custom adapter card signals

Pin number	Signal	Description
A1	/RBV	An active-low chip-select signal for the RAM-based video IC. This signal can be used, along with further decoding, to enable signals for a cache circuit.
A2	/FPU	An active-low chip-select signal for the Motorola MC68882 floating-point coprocessor. This device is decoded at a base address of $0002 2000 in the CPU address space (FC2 = FC1 = FC0 = 1). The A0 and SIZE pins of the 68882 are pulled high, signifying a 32-bit data bus.
B2	CACHE	An active-high signal asserted by an external cache to block the logic board memory controller from executing a memory cycle when the cache will provide the data. The timing of this signal must meet the requirements of the Macintosh IIci CACHE signal described in the section "Electrical Design Guidelines for the Cache Card" in Chapter 23.

Macintosh IIsi adapter card cache signals

If you are developing a cache circuit, further decoding logic must be added to derive two additional cache control signals, /CacheEnable and /CacheFlush. The decoding device can be a PAL, such as a 16L8B. Below are the equations for these two cache control signals.

/CacheEnable (low) = /RBV • /RW • /A0 • /A1 • /A4 • /D0
/CacheEnable (high) = /RBV • /RW • /A0 • /A1 • /A4 • D0 + RESET

/CacheFlush (low) = /RBV • /RW • /A0 • /A1 • /A4 • /D3
/CacheFlush (high) = /RBV • /RW • /A0 • /A1 • /A4 • D3

where • is Logical AND, + is Logical OR, and D0 is the logical complement of /D0.

All other combinations of inputs should not affect the /CacheEnable or /CacheFlush outputs. The /CacheEnable signal should be set low when writing a 0 to bit 0 of the RBV register 0, and should be set high when writing a 1 to bit 0 of RBV register 0, or when the RESET signal is asserted. The /CacheFlush signal should be set low when writing a 0 to bit 3 of the RBV register 0, and should be set high when writing a 1 to bit 3 of RBV register 0, or when the RESET signal is asserted.

Power consumption guidelines for the Macintosh IIsi adapter card

The allowable power consumption of an MC68030 processor-direct adapter card for the Macintosh IIsi is 2.5 W, including the MC68882 coprocessor. All of this power may be drawn from +5V.

The use of AC termination on the address, data, and control signals improves signal quality and is recommended by Apple. A series terminator with a 33 Ω resistor in series with a 120 pF capacitor to ground on all address, data, and control signals provides reduced noise.

Chapter 16 Electrical Design Guide for 68040 Direct Slot Expansion Cards

This chapter provides electrical guidelines for designing processor-direct expansion cards for the Macintosh Quadra 700 and Macintosh Quadra 900 computers. This section includes information on the following topics:

- electrical implementation of the 68040 Direct Slot
- functional description of expansion connector signals
- memory and I/O access for a 68040 expansion card
- interrupt processing
- power consumption guidelines

68040 Direct Slot expansion

The 68040 Direct Slot expansion connector, first used in the Macintosh Quadra 700 and the Macintosh Quadra 900, takes advantage of the more powerful MC68040 microprocessor. Like the 68030 Direct Slot expansion, the 68040 Direct Slot expansion supports 32-bit address and data buses. The pin count of this connector has been increased to 140 pins, as opposed to the 96 pins of the 68000 and 68020 PDS expansion connectors and the 120 pins of the 68030 PDS expansion connectors.

The pinouts of the expansion connectors used on the Macintosh Quadra 700 and the Macintosh Quadra 900 are identical. A PDS expansion card for the Macintosh Quadra family of computers must be designed to work with the 68040 microprocessor. PDS cards designed for computers that use the 68000, 68020, and 68030 will not work in the Macintosh Quadra 700 and Macintosh Quadra 900 computers.

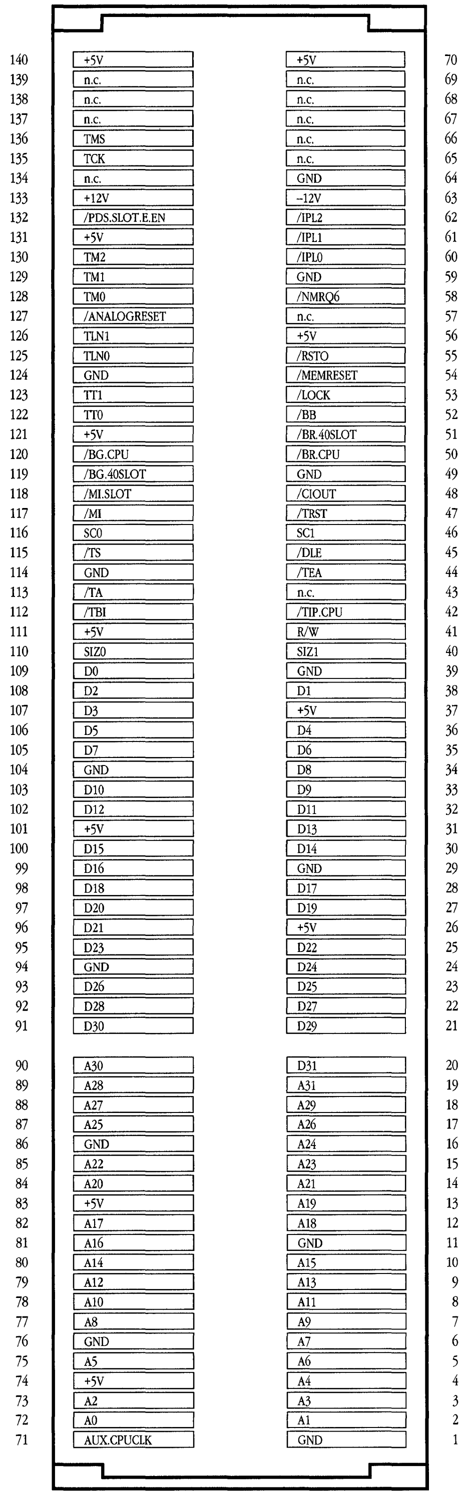
The following sections describe the pin assignments, define the signals, and provide signal load and drive information for the implementation of the 68040 Direct Slot on the Macintosh Quadra 700 and the Macintosh Quadra 900. This information is followed by two more sections that give specific design guidelines for PDS expansion cards in the Macintosh Quadra–family computers.

Electrical description of the 68040 Direct Slot

Figure 16-1 gives the pinout for the 140-pin expansion connector on the Macintosh Quadra 700 and Macintosh Quadra 900 main logic boards, as viewed from above.

Table 16-1 lists the pin assignments, gives the signal names, and briefly describes each signal. Table 16-2 shows two PDS signals that are connected to the microprocessor but must not be connected to a coprocessor on a PDS expansion card. Table 16-3 provides the load presented or drive available to each pin of an expansion card and indicates whether the signals are inputs or outputs.

■ **Figure 16-1** 68040 Direct Slot expansion connector pinout



140	+5V	+5V	70
139	n.c.	n.c.	69
138	n.c.	n.c.	68
137	n.c.	n.c.	67
136	TMS	n.c.	66
135	TCK	n.c.	65
134	n.c.	GND	64
133	+12V	–12V	63
132	/PDS.SLOT.E.EN	/IPL2	62
131	+5V	/IPL1	61
130	TM2	/IPL0	60
129	TM1	GND	59
128	TM0	/NMRQ6	58
127	/ANALOGRESET	n.c.	57
126	TLN1	+5V	56
125	TLN0	/RSTO	55
124	GND	/MEMRESET	54
123	TT1	/LOCK	53
122	TT0	/BB	52
121	+5V	/BR.40SLOT	51
120	/BG.CPU	/BR.CPU	50
119	/BG.40SLOT	GND	49
118	/MI.SLOT	/CIOUT	48
117	/MI	/TRST	47
116	SC0	SC1	46
115	/TS	/DLE	45
114	GND	/TEA	44
113	/TA	n.c.	43
112	/TBI	/TIP.CPU	42
111	+5V	R/W	41
110	SIZ0	SIZ1	40
109	D0	GND	39
108	D2	D1	38
107	D3	+5V	37
106	D5	D4	36
105	D7	D6	35
104	GND	D8	34
103	D10	D9	33
102	D12	D11	32
101	+5V	D13	31
100	D15	D14	30
99	D16	GND	29
98	D18	D17	28
97	D20	D19	27
96	D21	+5V	26
95	D23	D22	25
94	GND	D24	24
93	D26	D25	23
92	D28	D27	22
91	D30	D29	21
90	A30	D31	20
89	A28	A31	19
88	A27	A29	18
87	A25	A26	17
86	GND	A24	16
85	A22	A23	15
84	A20	A21	14
83	+5V	A19	13
82	A17	A18	12
81	A16	GND	11
80	A14	A15	10
79	A12	A13	9
78	A10	A11	8
77	A8	A9	7
76	GND	A7	6
75	A5	A6	5
74	+5V	A4	4
73	A2	A3	3
72	A0	A1	2
71	AUX.CPUCLK	GND	1

■ **Table 16-1** 68040 Direct Slot connector signals

Pin number	Signal name	Signal description
1	GND	Ground
2	A1	Address bit 1
3	A3	Address bit 3
4	A4	Address bit 4
5	A6	Address bit 6
6	A7	Address bit 7
7	A9	Address bit 9
8	A11	Address bit 11
9	A13	Address bit 13
10	A15	Address bit 15
11	GND	Ground
12	A18	Address bit 18
13	A19	Address bit 19
14	A21	Address bit 21
15	A23	Address bit 23
16	A24	Address bit 24
17	A26	Address bit 26
18	A29	Address bit 29
19	A31	Address bit 31
20	D31	Data bit 31
21	D29	Data bit 29
22	D27	Data bit 27
23	D25	Data bit 25
24	D24	Data bit 24
25	D22	Data bit 22
26	+5V	5 volts
27	D19	Data bit 19
28	D17	Data bit 17
29	GND	Ground
30	D14	Data bit 14
31	D13	Data bit 13
32	D11	Data bit 11
33	D9	Data bit 9

(continued)

■ **Table 16-1** 68040 Direct Slot connector signals (continued)

Pin number	Signal name	Signal description
34	D8	Data bit 8
35	D6	Data bit 6
36	D4	Data bit 4
37	+5V	5 volts
38	D1	Data bit 1
39	GND	Ground
40	SIZ1	Transfer size bit 1
41	R/W	Read/write
42	/TIP.CPU	Transfer in progress
43	n.c.	Not connected
44	/TEA	Transfer error acknowledge
45	/DLE	Data latch enable
46	SC1	Snoop control signal bit 1
47	/TRST	Test reset
48	/CIOUT	Cache inhibit out
49	GND	Ground
50	/BR.CPU	Bus request for main processor
51	/BR.40SLOT†	Bus request for PDS card
52	/BB	Bus busy
53	/LOCK	Bus lock
54	/MEMRESET†	Fast reset generated by JDB IC for Memory Control Unit
55	/RSTO	Reset out
56	+5V	5 volts
57	n.c.	Not connected
58	/NMRQ6†	NuBus slot $E interrupt; also connected to NuBus slot $E
59	GND	Ground
60	/IPL0	Interrupt priority 0
61	/IPL1	Interrupt priority 1
62	/IPL2	Interrupt priority 2
63	–12V	–12 volts
64	GND	Ground
65	n.c.	Not connected

(continued)

■ **Table 16-1** 68040 Direct Slot connector signals (continued)

Pin number	Signal name	Signal description
66	n.c.	Not connected
67	n.c.	Not connected
68	n.c.	Not connected
69	n.c.	Not connected
70	+5V	5 volts
71	AUX.CPUCLK†	Buffered version of main processor's bus clock
72	A0	Address bit 0
73	A2	Address bit 2
74	+5V	5 volts
75	A5	Address bit 5
76	GND	Ground
77	A8	Address bit 8
78	A10	Address bit 10
79	A12	Address bit 12
80	A14	Address bit 14
81	A16	Address bit 16
82	A17	Address bit 17
83	+5V	5 volts
84	A20	Address bit 20
85	A22	Address bit 22
86	GND	Ground
87	A25	Address bit 25
88	A27	Address bit 27
89	A28	Address bit 28
90	A30	Address bit 30
91	D30	Data bit 30
92	D28	Data bit 28
93	D26	Data bit 26
94	GND	Ground
95	D23	Data bit 23
96	D21	Data bit 21
97	D20	Data bit 20
98	D18	Data bit 18

(continued)

■ **Table 16-1** 68040 Direct Slot connector signals (continued)

Pin number	Signal name	Signal description
99	D16	Data bit 16
100	D15	Data bit 15
101	+5V	5 volts
102	D12	Data bit 12
103	D10	Data bit 10
104	GND	Ground
105	D7	Data bit 7
106	D5	Data bit 5
107	D3	Data bit 3
108	D2	Data bit 2
109	D0	Data bit 0
110	SIZ0	Transfer size bit 0
111	+5V	5 volts
112	/TBI	Transfer burst inhibit
113	/TA	Transfer acknowledge
114	GND	Ground
115	/TS	Transfer start
116	SC0	Snoop control signal bit 0
117	/MI	Memory inhibit
118	/MI.SLOT†	Memory inhibit from PDS card to Memory Control Unit
119	/BG.40SLOT†	Bus grant for PDS card
120	/BG.CPU	Bus grant for main processor
121	+5V	5 volts
122	TT0	Transfer type bit 0
123	TT1	Transfer type bit 1
124	GND	Ground
125	TLN0	Transfer line number bit 0
126	TLN1	Transfer line number bit 1
127	/ANALOGRESET†	Enables the PDS to drive the system reset signal; used only for testing
128	TM0	Transfer mode bit 0
129	TM1	Transfer mode bit 1
130	TM2	Transfer mode bit 2

(continued)

■ **Table 16-1** 68040 Direct Slot connector signals (continued)

Pin number	Signal name	Signal description
131	+5V	5 volts
132	/PDS.SLOT.E.EN†	Notifies YANCC that PDS card is installed and is using memory space assigned to NuBus slot $E
133	+12V	12 volts
134	n.c.	Not connected
135	TCK	Test clock
136	TMS	Test mode select
137	n.c.	Not connected
138	n.c.	Not connected
139	n.c.	Not connected
140	+5V	5 volts

† These signals are nonmicroprocessor signals on the Macintosh Quadra 700 and the Macintosh Quadra 900 and are not tied to any 68040 signal.

◆ *Note:* The AUX.CPUCLK line is terminated with a series resistor. To reduce reflections on this line, all loads on the card should be lumped.

■ **Table 16-2** Restricted 68040 microprocessor signals on the Macintosh Quadra 700 and Macintosh Quadra 900 PDS connectors

Signal name	Direction	Function
/TIP.CPU	Output	From the 68040 on the main circuit board; not connected to any other part of the computer
/IPL2–IPL0	Output	Interrupt priority lines from the PAL; not to be used as wire-OR lines; can be monitored by a PDS card

△ **Important** The signals on the Macintosh Quadra 700 and the Macintosh Quadra 900 PDS expansion connectors are connected directly to the 68040 with no buffers; that means the data and address buses and AUX.CPUCLK on a 68040 PDS expansion card must present capacitive loads of not more than 40 pF. All other lines must present capacitive loads of not more than 20 pF. △

■ **Table 16-3** 68040 Direct Slot signals, loading or driving limits

Signal name	Input/output	Load or drive limits
A0–A31	Input/Output	Load: 200 µA/2 mA, 150 pF Drive: 40 µA/.4 mA, 40 pF
D0–D31	Input/Output	Load: 200 µA/2 mA, 120 pF Drive: 40 µA/.4 mA, 40 pF
/ANALOGRESET	Output	Drive: 40 µA/.4 mA, 20 pF
AUX.CPUCLK	Output	Drive: 40 µA/.4 mA, 40 pF
/BB	Input/Output	Load: 100µA/4 mA, 70 pF Drive: 40 µA/.4 mA, 20 pF
/BG.CPU	Output	Drive: 40 µA/.4 mA, 20 pF
/BG.40SLOT	Output	Drive: 40 µA/.4 mA, 20 pF
/BR.CPU	Input	Load: 100 µA/4 mA, 40 pF
/BR.40SLOT	Input	Load: 100 µA/4 mA, 40 pF
/CIOUT	Output	Drive: 40 µA/.4 mA, 20 pF
/DLE	Input	Load: 100 µA/4 mA, 25 pF
/IPL0–/IPL2	Output	Drive: 40 µA/.4 mA, 20 pF
/LOCK	Input/Output	Load: 100 µA/4 mA, 25 pF Drive: 40 µA/.4 mA, 20 pF
/MEMRESET	Output	Drive: 40 µA/.4 mA, 20 pF
/MI	Input/Output	Load: 100 µA/4 mA, 25 pF Drive: 40 µA/.4 mA, 40 pF
/MI.SLOT	Input	Load: 100 µA/4 mA, 20 pF
/NMRQ6	Input	Load: 100 µA/4 mA, 40 pF
/PDS.SLOT.E.EN	Input	Load: 100 µA/4 mA, 20 pF
/RSTO	Output	Drive: 40 µA/.4 mA, 20 pF
R/W	Input/Output	Load: 100 µA/4 mA, 100 pF Drive: 40 µA/.4 mA, 20 pF
SC0–SC1	Input/Output	Load: 100 µA/4 mA, 40 pF Drive: 40 µA/.4 mA, 20 pF
SIZ0–SIZ1	Input/Output	Load: 100 µA/4 mA, 90 pF Drive: 40 µA/.4 mA, 20 pF
/TA	Input/Output	Load: 100 µA/4 mA, 100 pF Drive: 40 µA/.4 mA, 20 pF

(continued)

■ **Table 16-3** 68040 Direct Slot signals, loading or driving limits (continued)

Signal name	Input/output	Load or drive limits
/TBI	Input/Output	Load: 100 µA/4 mA, 100 pF Drive: 40 µA/.4 mA, 20 pF
TCK	Input	Load: 100 µA/4 mA, 25 pF
/TEA	Input/Output	Load: 100 µA/4 mA, 100 pF Drive: 40 µA/.4 mA, 20 pF
/TIP.CPU	Output	Drive: 40 µA/.4 mA, 20 pF
TLN0–TLN1	Output	Drive: 40 µA/.4 mA, 20 pF
TM0–TM2	Output	Drive: 40 µA/.4 mA, 20 pF
TMS	Input	Load: 100 µA/4 mA, 25 pF
/TRST	Input	Load: 100 µA/4 mA, 25 pF
/TS	Input/Output	Load: 100 µA/4 mA, 90 pF Drive: 40 µA/.4 mA, 20 pF
TT0–TT1	Input/Output	Load: 100 µA/4 mA, 90 pF Drive: 40 µA/.4 mA, 20 pF

◆ *Note:* Input denotes direction from PDS card to the main logic board, not necessarily to the processor. Output denotes direction from main logic board, and not necessarily from the processor, to the PDS card.

Design considerations for 68040 Direct Slot expansion cards

The following paragraphs provide information that you should become familiar with before starting your expansion card design. Included are a description of how an expansion card gains access to memory and I/O devices, a description of how the interrupt-handling mechanism works, a summary of design hints, and a discussion of the expansion card power requirements for the Macintosh Quadra 700 and the Macintosh Quadra 900.

Bus master priority scheme

For maximum performance, the processor-direct slot is connected directly to the 68040 microprocessor by way of the system bus. There are actually three buses in the Macintosh Quadra 700 and the Macintosh Quadra 900: the system bus, the I/O bus, and the NuBus.

The system bus connects directly to the pins of the 68040 microprocessor and runs at the processor's clock rate, 25 MHz. Five types of controller ICs are connected to the system bus: the Memory Control Unit, a custom memory controller; YANCC, the NuBus controller; DAFB, the frame buffer controller; the I/O adapter chips; and the SCSI controllers.

The I/O bus in the Macintosh Quadra–family computers is similar to the I/O bus in the Macintosh IIfx and runs at a clock rate of 15.6672 MHz. The controller ICs that are connected to the I/O bus include the Enhanced Apple Sound chip and the Sonic custom IC as well as ICs shared with older Macintosh models.

Finally, the NuBus runs at a clock rate of 10 MHz and supports any NuBus expansion cards. For more information about the NuBus expansion interface for the Macintosh Quadra 700 and the Macintosh Quadra 900, refer to Part I.

It is possible to have multiple bus masters on the Macintosh IIfx processor bus. The possible bus masters and their position in the priority scheme are shown in Table 15-14. Note that the NuBus and SCSI interfaces allow DMA access to the 68030 Direct Slot.

The Macintosh Quadra–family computers can support up to four bus masters: three on the system bus and one on the I/O bus. The Relayer ASIC contains the bus arbitration logic. The possible bus masters and their position in the priority scheme are shown in Table 16-4. The arbitration mechanism includes a degree of fairness that should keep devices from becoming bus-starved. Keep in mind that the current bus master has complete control of both the system bus and the I/O bus. For example, if the 68040 is the current bus master and the Sonic Ethernet controller requests the bus, the Ethernet controller will have to wait until the 68040 relinquishes control before it can drive the bus.

■ **Table 16-4** Bus master priority scheme for the Macintosh Quadra–family computers

Priority	Bus	Device
First (highest)	I/O bus	Sonic Ethernet controller
Second	System bus	YANCC NuBus controller
Third	System bus	PDS
Fourth (lowest)	System bus	68040

The 68040 is the lowest-priority device, but the arbiter does support bus parking, so the average latency for 68040 bus access is minimized. If a system bus master attempts to burst data to or from an I/O bus slave, the Relayer ASIC will invoke a 68040 fake burst by asserting the 68040 /TBI signal.

Memory and I/O access for expansion cards

A PDS card can have memory locations in the upper part of the RAM memory space ($1000 0000 through $3FFF FFFF) or in the space assigned to NuBus slot $E ($FE00 0000 through $FEFF FFFF or $E000 0000 through $EFFF FFFF). See Table 16-5 for a listing of the memory map physical address space assignments for the computers in the Macintosh Quadra family. If you are designing a new processor-direct expansion card for the Macintosh Quadra 700 or the Macintosh Quadra 900, you should use the pseudoslot design method to emulate this NuBus address space. Pseudoslot design is explained in the next section. The advantage of designing a PDS card to occupy one of the unused NuBus addresses is that existing ROM firmware, which has the ability to manage NuBus slots, is present in the system ROM. If you design your card along the lines of a NuBus card (with a declaration ROM and interrupt capability), the Slot Manager in ROM controls your card as if it were a NuBus card, but the electrical interface is connected directly to the 68040 processor.

The Memory Control Unit (MCU) IC connects to the system bus and provides control and timing signals for RAM and ROM on the Macintosh Quadra 700 and the Macintosh Quadra 900. Among the features of the 68040 is the ability to mark different areas in memory—both RAM and ROM—as cacheable or noncacheable. The MCU supports all types of 68040 memory access, including burst modes.

■ **Table 16-5** Macintosh Quadra 700 and Macintosh Quadra 900 32-bit physical address spaces

Address	**Description**
$0000 0000–$3FFF FFFF	RAM
$4000 0000–$4FFF FFFF	ROM
$5000 0000–$5000 1FFF	VIA1
$5000 2000–$5000 3FFF	VIA2
$5000 4000–$5000 7FFF	Reserved for Apple
$5000 8000–$5000 9FFF	Ethernet PROM
$5000 A000–$5000 BFFF	Ethernet
$5000 C000–$5000 DFFF	IOP for SCC (Macintosh Quadra 900); SCC (Macintosh Quadra 700)
$5000 E000–$5000 EFFF	Memory Control Unit controls
$5000 F000–$5000 F3FF	SCSI 0 (internal)
$5000 F400–$5000 F7FF	SCSI 1 (external) in the Macintosh Quadra 900; reserved for Apple in the Macintosh Quadra 700
$5000 F800–$5000 3FFF	Reserved
$5001 4000–$5001 5FFF	Sound
$5001 6000–$5001 DFFF	Reserved for Apple
$5001 E000–$5001 FFFF	IOP for SWIM and ADB (Macintosh Quadra 900); SWIM (Macintosh Quadra 700)
$5002 0000–$5002 7FFF	Reserved for Apple
$5002 8000–$5002 9FFF	YANCC controls
$5002 A000–$5003 FFFF	Reserved for Apple
$5004 0000–$53FF FFFF	Reserved (duplicate images of I/O space $5000 0000–$5004 0000)
$5400 0000–$5FFF FFFF	Reserved for Apple
$6000 0000–$EFFF FFFF	NuBus super slots
$F000 0000–$F0FF FFFF	Reserved
$F100 0000–$FFFF FFFF	NuBus slots

Pseudoslot design guidelines for PDS expansion cards

If you are familiar with designing devices for the MC68000 family of microprocessors, you should find it relatively easy to use the pseudoslot method to design an expansion card for the Macintosh Quadra 700 and the Macintosh Quadra 900. The only added constraints are that you need a declaration ROM and that you must adhere to some address decoding rules.

Many of the address locations correspond to address ranges used by NuBus expansion cards resident in Macintosh computers that offer NuBus. The advantage of designing an expansion card to occupy one of these unused addresses is that existing ROM firmware with the ability to manage the NuBus slots is also present in your computer's system ROM. Therefore, if an expansion card is designed along the lines of a NuBus card (for example, with a declaration ROM and interrupt capability), the existing Slot Manager ROM firmware controls this card as if it were a NuBus card, but the electrical interface is via the 68040 bus. As a side benefit of this design, one software driver works on machines with two different methods of expansion. To find out more about the Slot Manager, refer to Chapter 8, "NuBus Card Firmware," in this book and the Slot Manager information in *Inside Macintosh.*

A PDS card in a Macintosh Quadra–family computer can have memory locations in the upper part of the RAM memory space or in the space assigned to NuBus slot $E. If the card uses slot $E addresses, it must decode all addresses in both the slot space and the super slot space, responding to any access to an unused location with /TEA on the processor bus to indicate an illegal address.

A typical PDS card maps into the NuBus space and works with the system software's Slot Manager. Such a card must notify the NuBus controller that it is using the NuBus space so that the NuBus controller will ignore accesses to slot $E. To do that, the card asserts the signal /PDS.SLOT.E.EN on the PDS connector by pulling the line low.

A PDS card that asserts the /PDS.SLOT.E.EN signal must issue a /TA (transfer acknowledge) or a /TEA in response to all accesses to the $Exxx xxxx and $FExx xxxx address space. (This action will keep the machine from hanging since there is no time-out timer for slot $E when the /PDS.SLOT.E.EN signal is asserted.)

A PDS bus master card must drive all control signals to a known state when it requests the system bus. Snoop control bits, in particular, must be driven to indicate no snoop.

By convention, all devices on the system bus, including a PDS card, must drive tristate signals active for one-half of a clock cycle before going tristate. For example, a PDS card functioning as the slave should drive /TA high after the address space is decoded, then drive it low for one clock cycle, and finally drive it high at the end of that clock cycle. The /TA signal should go tristate one-half clock cycle later. If you follow these guidelines in your PDS card design, the control lines have cleaner edges and so your card operates more reliably.

The declaration ROM must reside at the upper address limit of the 16 MB address space in order for the Slot Manager code to recognize the card. Chapter 8, "NuBus Card Firmware," provides information to help you develop the necessary card firmware.

You are not required to follow the pseudoslot method for designing an I/O expansion card. This method is provided as a way of simplifying the design task and minimizing the need for revisions of support software.

Timing considerations

Signal timing for the PDS connector is dependent on the clock speed of the 68040 microprocessor. Developers of PDS expansion cards should clearly indicate on the card the maximum clock speed of the card.

The timing of a PDS card's output signals must be equal to or better than the worst-case delay output timing of the 68040 microprocessor. Also, input signals to a PDS card cannot expect more setup time than is required for a signal to set up on the 68040 microprocessor.

68040 Direct Slot interrupt handling

The interrupt-handling mechanism for the 68040 Direct Slot on the Macintosh Quadra 700 and the Macintosh Quadra 900 is similar to the one used in Macintosh computers with NuBus. There are five NuBus slot interrupt signals on the Macintosh Quadra 900 and two on the Macintosh Quadra 700. The interrupt signal for slot $E is shared by the 68040 Direct Slot. The NuBus slot interrupt signals, the on-board video interrupt signal, and the Ethernet controller interrupt signal are routed through an OR gate to generate a signal called /SLOTIRQ. This signal is connected to the CA1 input of VIA2, the second VIA chip on the logic board. This VIA generates a level 2 interrupt to the 68040. This VIA can also generate an interrupt in response to SCSI requests, sound chip requests, or VIA timer requests.

All interrupts to the 68040 are autovectored. When the 68040 is executing a level *x* interrupt, it first sets the interrupt mask to level *x*, so further interrupts at level *x* and below will be ignored. Once the interrupt handler is executed and an RTE instruction is processed, the interrupt mask is restored to the value it had before the interrupt.

The first-level interrupt dispatcher determines which hardware device—SCSI, sound chip, real-time clock, or expansion slot—is requesting the interrupt and dispatches code to the appropriate interrupt handler. If the interrupt generated by the VIA is a slot interrupt, the software polls the second VIA, bits PA0 through PA6, to determine which slot generated the interrupt. Table 16-6 summarizes the interrupt lines for VIA2.

■ **Table 16-6** Macintosh Quadra 700 and Macintosh Quadra 900 VIA2 interrupt lines

Address	Description
PA0	Ethernet IRQ
PA1†	Slot $A IRQ
PA2†	Slot $B IRQ
PA3†	Slot $C IRQ
PA4	Slot $D IRQ
PA5	Slot $E IRQ
PA6	Video IRQ

† The PA1, PA2, and PA3 interrupt lines on the Macintosh Quadra 700 are not connected.

Once the software determines which pseudoslot generated the interrupt, the Slot Manager software executes the interrupt handler for that slot device. The handler for that device was installed at boot time, when the initialization software polled the possible slots and identified the existence of a card in the slot by its ROM signature.

There is a delay between the assertion of a slot interrupt and the actual execution of the interrupt handler. During this time, the software polls the actual slot /IRQ signal. The recommended design practice is to latch the slot /IRQ signal so that once it is asserted, the interrupt-handler software for the card has the responsibility of clearing the interrupt. This ensures that the slot /IRQ signal is asserted when polled and the Slot Manager is dispatched correctly.

In addition to the standard Macintosh II functions, the VIA1 includes 2 new bits. The first is a software interrupt signal, and the second is the A/UX interrupt enable signal. When the software interrupt bit is set, an interrupt will be passed to the 68040. When the A/UX interrupt enable bit is set, the interrupt control PAL will remap the interrupts. Table 16-7 shows the Macintosh and A/UX operating-system interrupts, where priority 0 is low and priority 7 is high.

■ **Table 16-7** Macintosh Quadra 700 and Macintosh Quadra 900 interrupt mapping

Interrupt priority	Macintosh II interrupt	A/UX interrupt
0	—	—
1	VIA1	Software
2	VIA2 (SCSI, sound, NuBus slots, Ethernet, video)	VIA2 (SCSI, NuBus slots, video)
3	—	Ethernet
4	SCC	SCC
5	—	Sound
6	—	VIA1
7	NMI/YANCC error	NMI/YANCC error

Cache management

The 68040 microprocessor has two internal caches, one for instructions and one for data. The caches perform the same function as those on earlier processors, storing the contents of recently addressed memory locations in anticipation that those contents will soon be used again.

The data cache in the 68040 microprocessor has a new mode called CopyBack mode. That mode is different from the WriteThru mode used by the caches in the MC68020 and MC68030 microprocessors. CopyBack mode improves the overall performance because the processor may write to a memory location several times before the data must be flushed from the cache. Operating in CopyBack mode can increase the processor's performance by up to 50% but also requires the operating system to manage some types of data more carefully.

The difference between the WriteThru and CopyBack modes on the 68040 processor is the way they deal with data being written to memory. In WriteThru mode, the 68040 writes the data to the cache and also updates main memory immediately. In CopyBack mode, the 68040 writes directly to the cache; main memory is not immediately updated. The cache writes the data to main memory when that portion of the data is selected for replacement or when the data cache is flushed.

Cache management by ROM

One consequence of the use of CopyBack mode is that main memory does not always contain the latest data. There could be a problem when an alternate bus master reads from memory that is being cached by the main processor and the main memory has not been updated. To prevent this problem from arising, the ROM software uses only pages marked uncacheable when setting up communication areas with alternate bus masters.

Another way old data can cause a problem is when the microprocessor fills its instruction cache from an area with old data. To prevent that problem, the ROM software flushes the contents of the data cache to main memory after writing data that consists of instruction code. Specifically, the software flushes the data cache to main memory after each of the following operations:

- loading a resource into memory
- moving a heap block
- creating a jump table

◆ *Note:* The ROM software in the Macintosh Quadra–family computers also invalidates the caches in the 68040 in the same places that the older ROM software invalidated the caches in the 68020 and 68030.

Cache management by applications

It has always been important to flush the caches on the 68020 and 68030 before executing instructions that were recently written to memory. On the 68040, flushing only the instruction cache in this situation is not sufficient. The instruction and data caches are independent of each other, and there is a strong possibility that the instruction cache will fill with old data from RAM while the new data has not yet been written to RAM from the data cache.

To prevent this problem in your applications, you must use one or more of the calls provided by the system software whenever you write data that will be executed as instructions. You should use the `_FlushInstructionCache` and `_FlushDataCache` calls to flush each cache. Because the purpose of the `_FlushInstructionCache` call is to maintain cache coherency, its operation on the 68040 is to flush both the instruction and data caches. Flushing both caches with one call also avoids problems in situations where interrupts might occur while the caches are being flushed individually.

For more information about the `_FlushInstructionCache` and `_FlushDataCache` function calls, contact Macintosh Developer Technical Support (MacDTS). In the future, these function calls will be documented in *Inside Macintosh.*

△ **Important** Flushing the cache at certain times is critically important, but don't flush the cache too often. Unnecessary flushing of the cache impairs the performance of the 68040 microprocessor. △

Design hints for PDS expansion cards in Macintosh Quadra–family computers

When designing a card for the Macintosh Quadra 700 and the Macintosh Quadra 900, you must generate timing to match the requirements of the 68040 microprocessor. For further information on the timing requirements of the microprocessor, refer to the Motorola *MC68040 32-Bit Microprocessor User's Manual.*

The 68040 also has a new feature called bus **snooping.** Snooping is a hardware function that allows the cache to monitor the bus activity by alternate bus masters. The snooping function is a necessary part of cache coherency. Because snooping substantially slows the processor down, however, it is not used in the Macintosh Quadra 700 and the Macintosh Quadra 900 and is not supported by the software. Devices that transfer data on the system bus, such as PDS bus masters, must drive the snoop control pins (SC0 and SC1) to indicate no snooping. Because snooping is not supported, cache coherency must be maintained explicitly. Refer to the previous section for information about maintaining cache coherency.

Starting with the Macintosh IIci computer, ROM software has provided some of the virtual memory (VM) routines to allow programmers to manipulate the tables in the MMU. The ROM software for the Macintosh Quadra–family computers includes modifications to those routines to support the 68040, along with some enhancements to provide write-protection capability for the main memory.

Specifically, the existing routines `LockMemory`, `LockMemoryContiguous`, and `UnlockMemory` can change the attributes of individual pages in the absence of VM. The ability to set the attributes of individual pages in memory becomes important with the advent of the large internal caches of the 68040 and the common use of alternate bus masters. For example, when an area in memory is used as a communication buffer between the main processor and an alternate bus master, that area of memory must be marked uncacheable to maintain cache coherency after writes from the alternate bus master. On earlier machines that used the 68020 and 68030 processors, it was acceptable to turn off the entire cache whenever any pages needed to be uncacheable, due to the small sizes of the caches on those processors and the limited number of bus masters. On a machine with a 68040 and on-board bus masters, such a practice would result in an unacceptable degradation of performance.

Power consumption guidelines for 68040 Direct Slot expansion cards

The power budget for a PDS expansion card in the Macintosh Quadra 700 and Macintosh Quadra 900 is identical to the power budget for the NuBus card that it replaces. Refer to the section "NuBus Power Budget" in Chapter 5 for details.

Chapter 17 Physical Design Guide for Macintosh PDS Expansion Cards

This chapter contains physical design guidelines for developing expansion cards for Macintosh computers with processor-direct slots. Included in

this category are the Macintosh SE (68000 Direct Slot), the Macintosh Portable (68000 Direct Slot), the Macintosh LC (68020 Direct Slot), the Macintosh SE/30 (68030 Direct Slot), the Macintosh IIsi (68030 Direct Slot), the Macintosh IIfx (68030 Direct Slot), and the Macintosh Quadra 700 and Macintosh Quadra 900 (68040 Direct Slot) computers.

This chapter includes

- mechanical drawings showing expansion card dimensions and mounting provisions
- descriptions of the mating 96-pin, 120-pin, and 140-pin connectors on the expansion cards and logic boards
- mechanical drawings detailing electrical and physical requirements for connecting expansion cards to external equipment

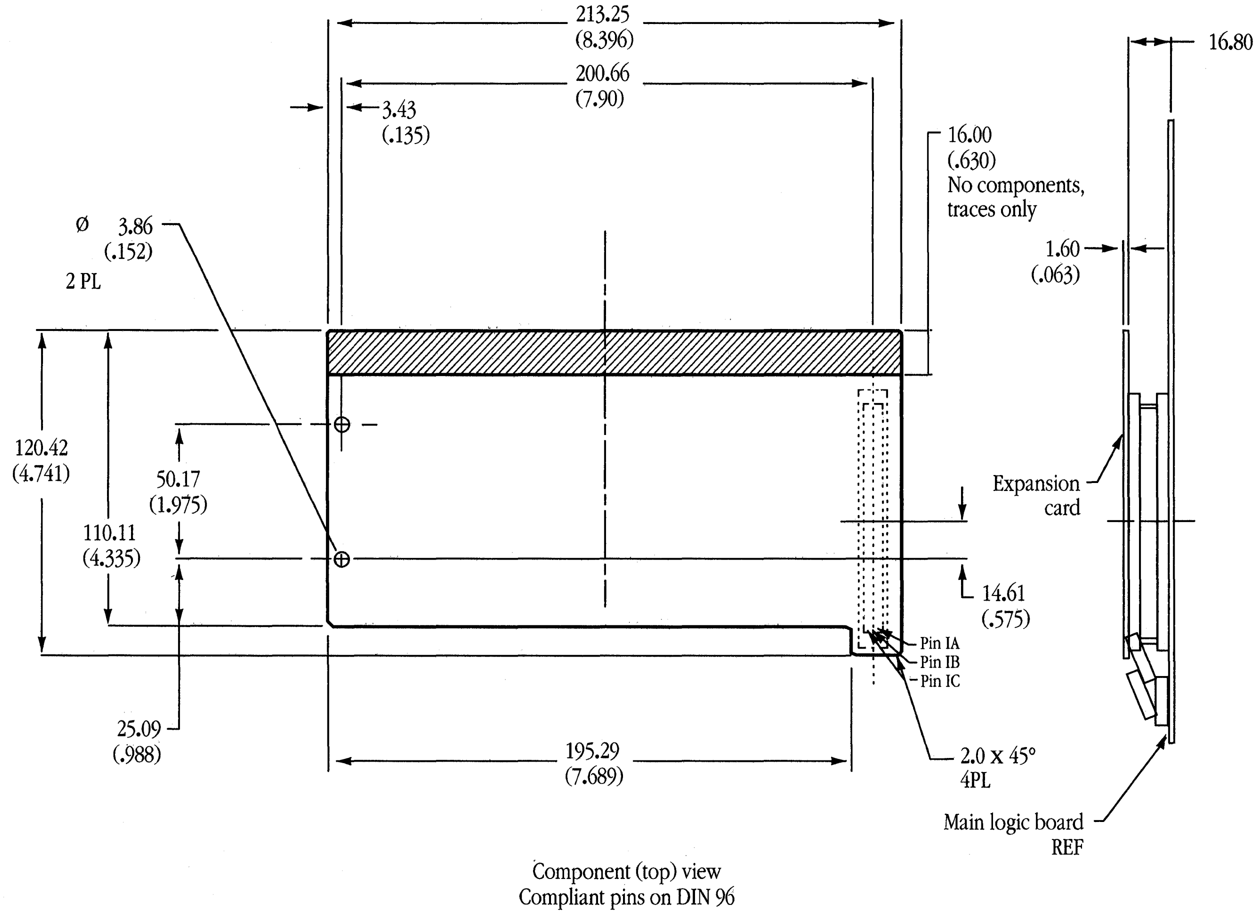
▲ **Warning** The drawings in this chapter are from mechanical design guides used within Apple Computer. They were correct at the time of publication but are subject to change. ▲

Physical guidelines for Macintosh SE expansion cards

Figures 17-1 through 17-3 show the spatial relationship between an expansion card and the Macintosh SE main logic board.

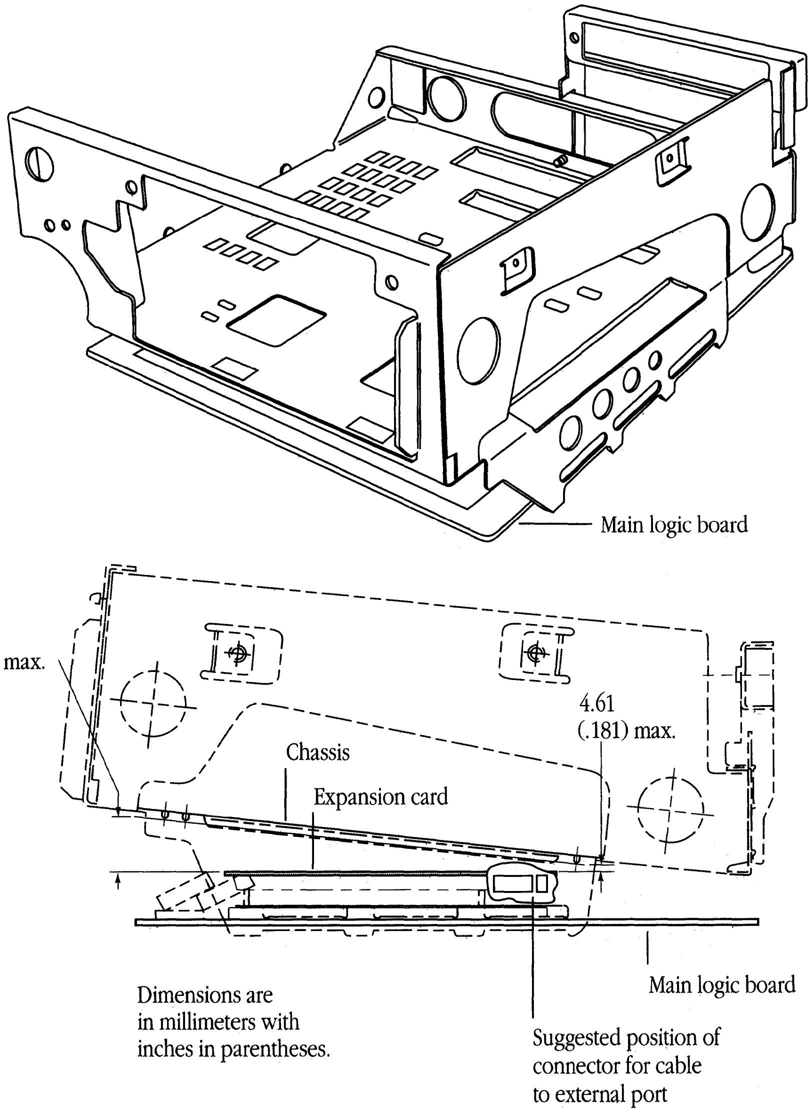
▲ **Warning** Figure 17-1 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change. ▲

■ **Figure 17-1** Macintosh SE expansion card design guide



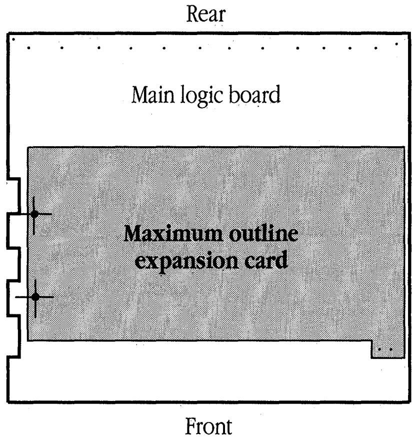
Dimensions are in millimeters with inches in parentheses.

■ **Figure 17-2** An expansion card in the Macintosh SE assembly



Side view height restrictions

■ **Figure 17-3** An expansion card and the Macintosh SE main logic board



The 68000 Direct Slot 96-pin connector for the Macintosh SE

Figure 17-4 shows a plug connector that mates with the Euro-DIN 96-pin socket connector on the main logic board. The plug connector should have compliant pins (force fit insertion) rather than solder-type pins for connection to the expansion card if components are to be mounted on the top side of the card.

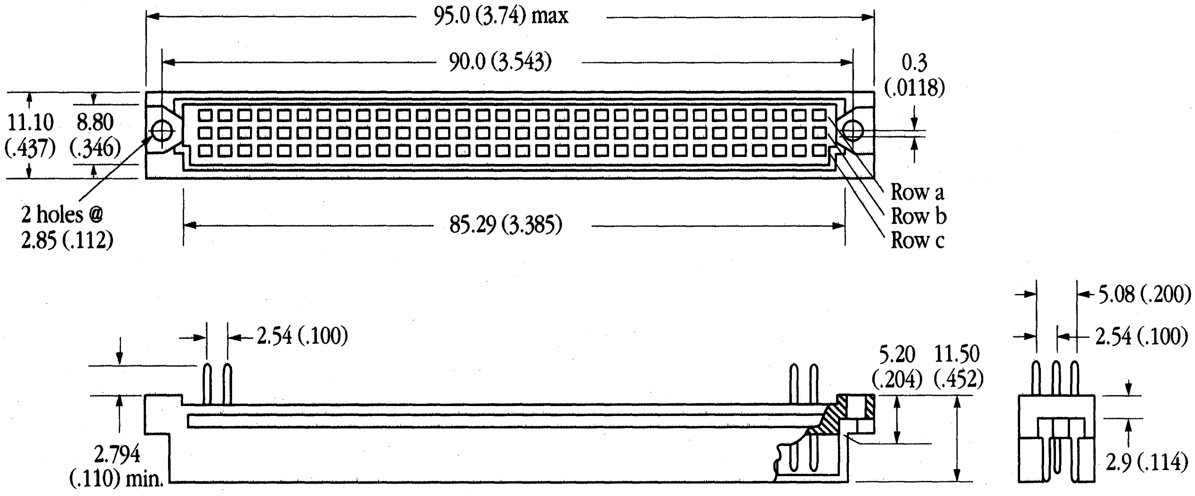
Figure 17-5 shows the 96-pin socket connector and mounting supports on the Macintosh SE main logic board assembly. Figure 17-6 is a detail of the socket connector used on the main logic board.

You can order Euro-DIN 96-pin connectors meeting Apple specifications from

AMP Incorporated
Harrisburg, PA 17105

Because of high-volume production requirements, Apple purchases specially modified versions of the Euro-DIN 96-pin connector from this vendor. However, you may purchase a mating connector of standard configuration from this or other vendors.

■ **Figure 17-4** A 96-pin plug connector for a Macintosh SE expansion card

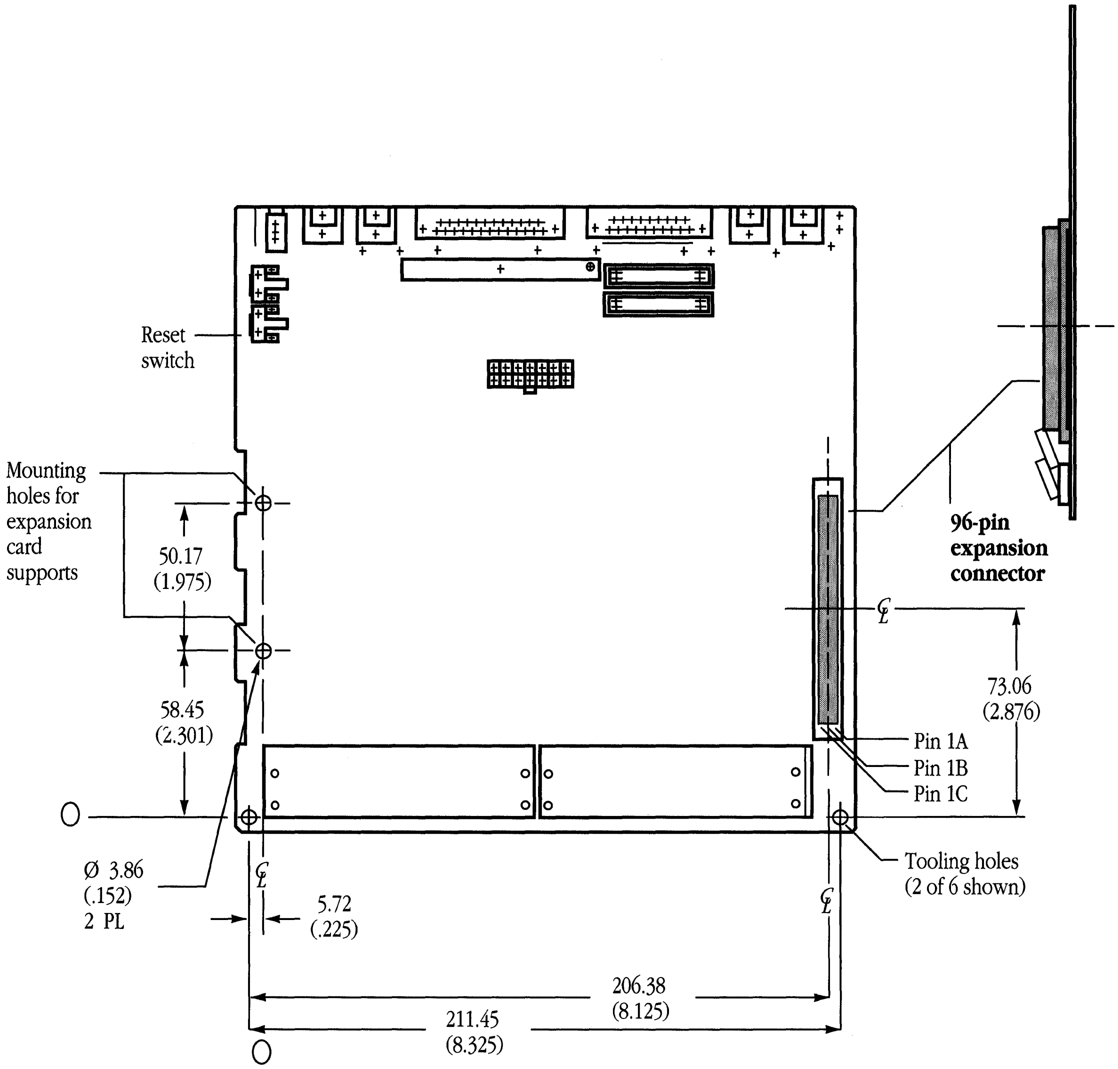


Three-row pin connector

96 contact positions
2.54 mm (.100 inch) spacing pins
Gold plated, 20 microinches, over nickel plate

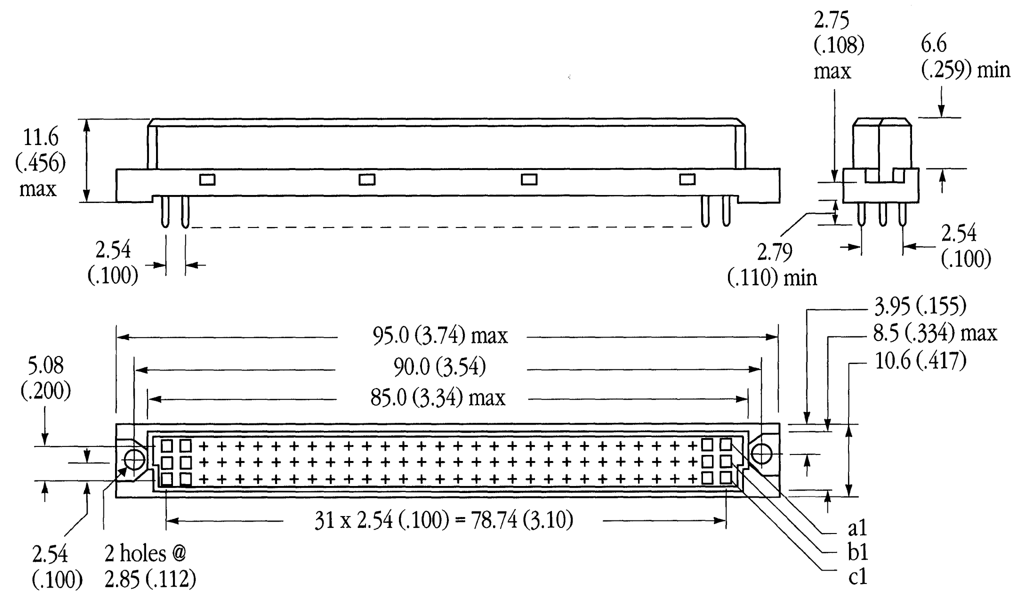
Dimensions are in millimeters with inches in parentheses.

■ **Figure 17-5** Macintosh SE connector and mounting supports for an expansion card



Dimensions are in millimeters with inches in parentheses.

■ **Figure 17-6** Detail of 96-pin socket connector used on Macintosh SE main logic board



Three-row socket connector

96 contact positions
2.54 mm (.100 inch) spacing sockets
Gold plated, 20 microinches, over nickel plate

Dimensions are in millimeters with inches in parentheses.

External connections for the Macintosh SE

This section discusses both electrical and physical considerations required in making connections to external equipment for the Macintosh SE.

The Macintosh SE has an external device access opening through which another piece of equipment can be connected. Typically, a cable would be routed from the expansion card upward through a cutout in the back of the chassis, and then to a connector on a connector card you provide.

Mechanical drawings in this section show the provision Apple has made for connecting your expansion card to devices external to the Macintosh SE.

◆ *Note:* FCC regulations on radio-frequency emissions prohibit the installation of an external device access opening in the Macintosh Portable computer.

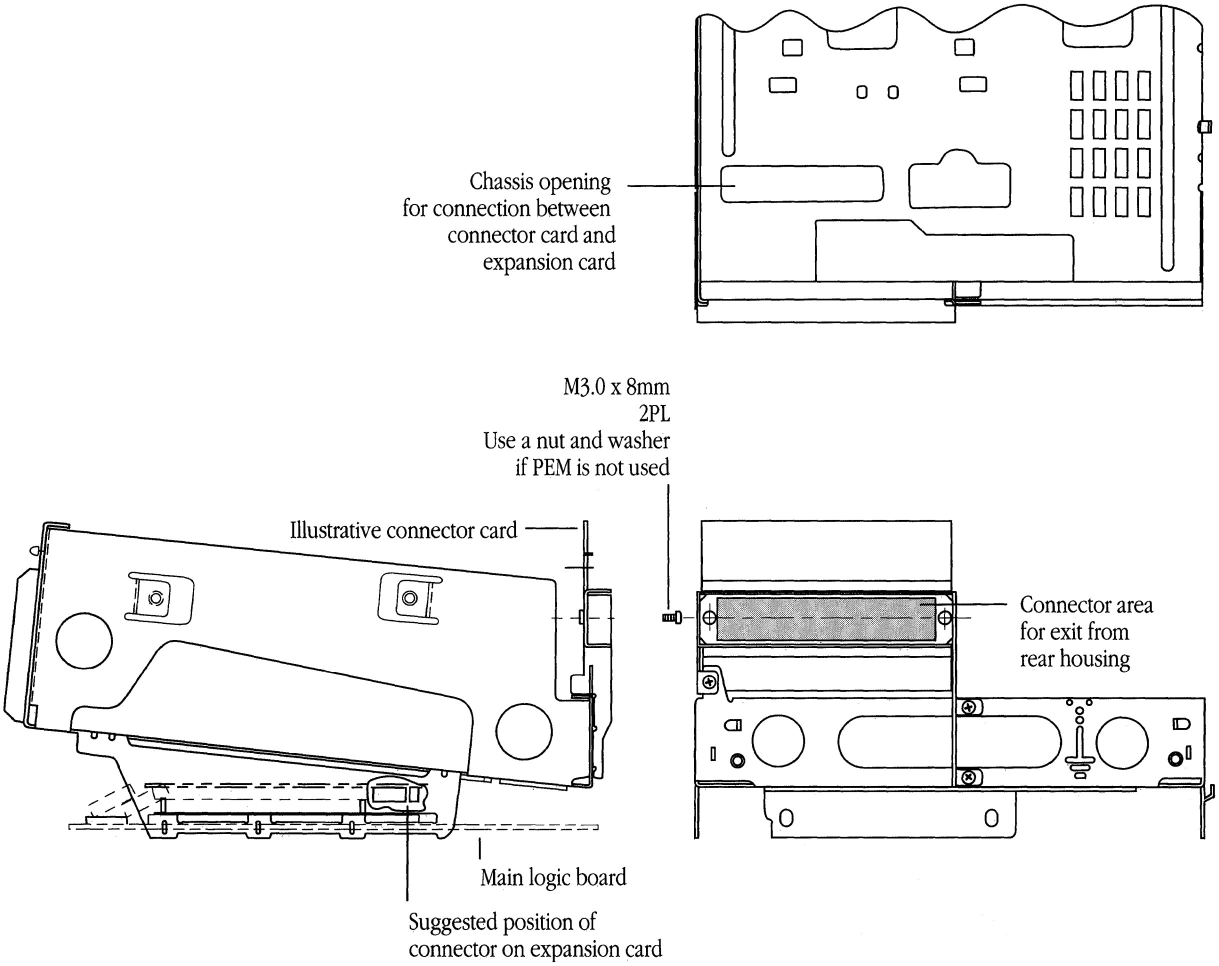
Figure 17-7 shows the Macintosh SE sheet metal and the structure for mounting your connector and connector card. Figure 17-8 shows the recommended internal cable routing paths for the Macintosh SE.

△ **Important** If you design your expansion card's internal cable to be at least 220 mm (8.6 inches) long, it can also be used to connect an expansion card to the Macintosh SE/30, the Macintosh IIsi, and other future Macintosh computers. △

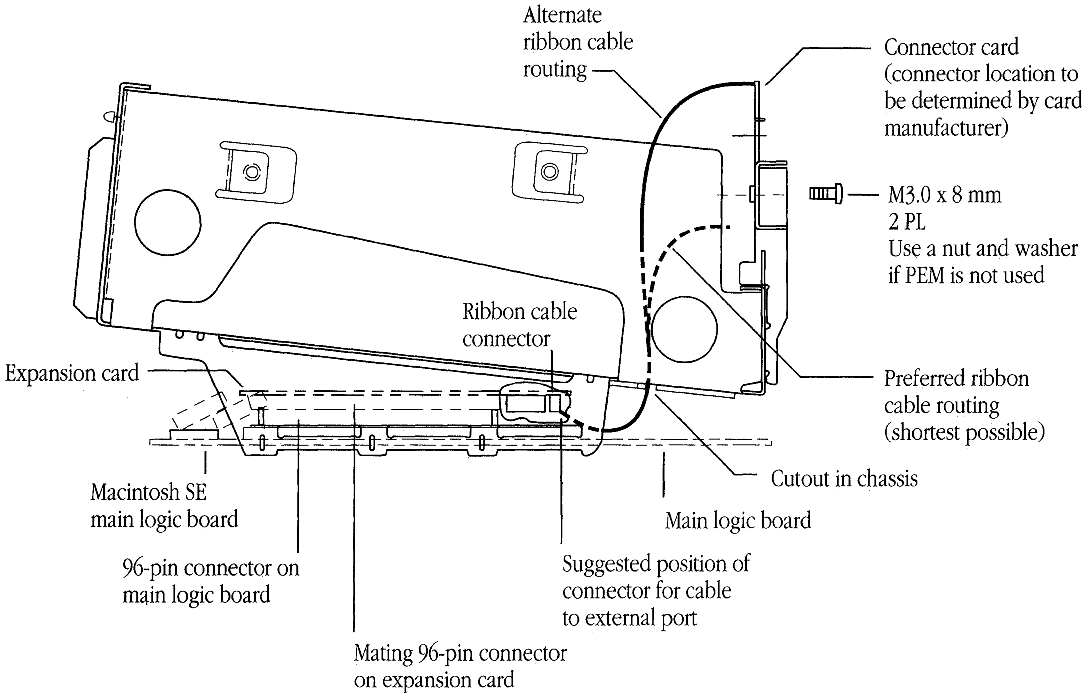
Foldout 8 at the end of the book is a design guide for the connector card. All areas of significant importance are noted on the drawing. If you design a connector card that adheres to the dimensions in Foldout 8, it can be used on the Macintosh SE, the Macintosh SE/30, the Macintosh IIsi, and future versions of the compact Macintosh.

▲ **Warning** Foldout 8 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change. ▲

■ **Figure 17-7** Connector card mounting on Macintosh SE chassis



■ **Figure 17-8** Internal expansion cable routing for Macintosh SE

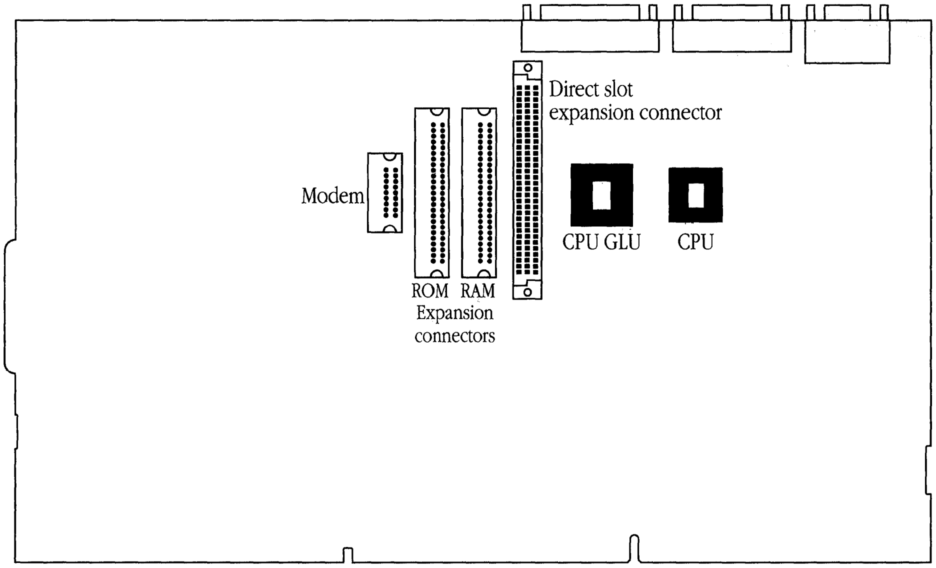

Alternate ribbon cable routing
Connector card (connector location to be determined by card manufacturer)
M3.0 x 8 mm
2 PL
Use a nut and washer if PEM is not used
Ribbon cable connector
Expansion card
Preferred ribbon cable routing (shortest possible)
Cutout in chassis
Macintosh SE main logic board
Main logic board
96-pin connector on main logic board
Suggested position of connector for cable to external port
Mating 96-pin connector on expansion card

Physical guidelines for Macintosh Portable expansion cards

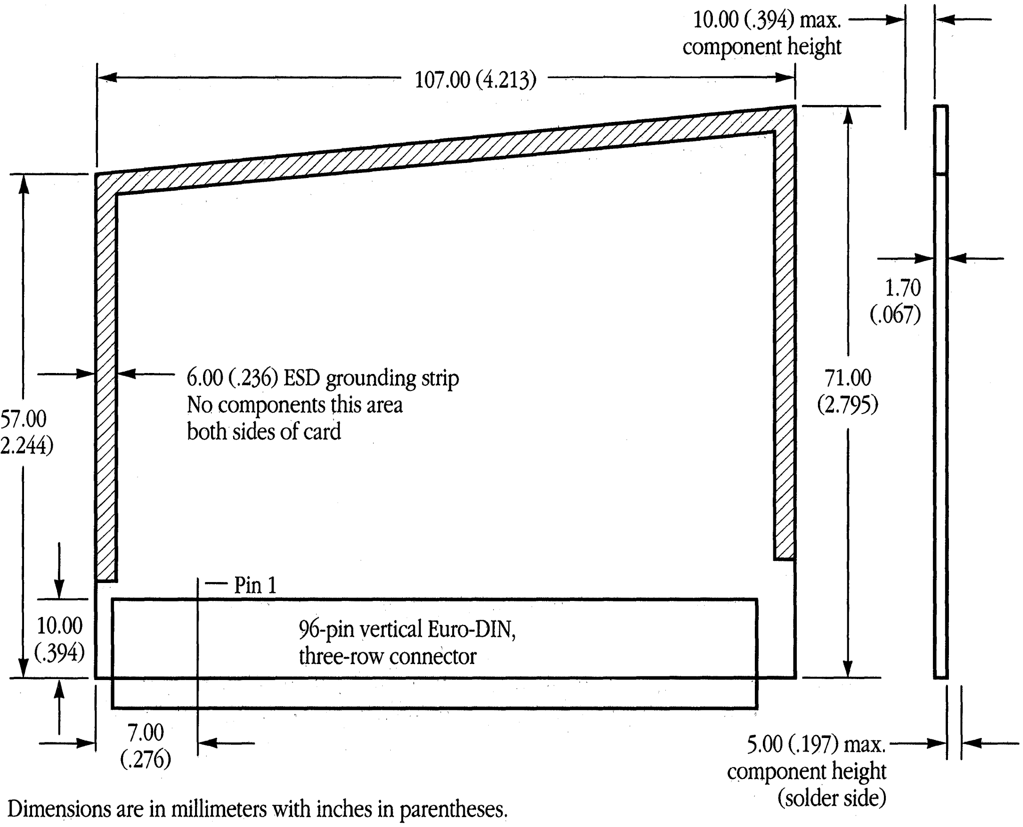
Figure 17-9 shows the location of the 96-pin expansion connector on the Macintosh Portable main logic board. Figure 17-10 is a design guide showing the size of the card, the location of the 96-pin connector, and the maximum allowable component mounting height. The Macintosh Portable uses the same Euro-DIN 96-pin expansion connector described earlier in this chapter in the section "The 68000 Direct Slot 96-Pin Connector for the Macintosh SE." The connectors for the expansion card and the main logic board are the same as those shown for the Macintosh SE in Figures 17-4 and 17-6.

◆ *Note:* Before designing an expansion card for the Macintosh Portable, make sure you are aware of the limitations described in the section "68000 Direct Slot Expansion for the Macintosh Portable" in Chapter 13.

■ **Figure 17-9** Expansion connector location on Macintosh Portable main logic board



- **Figure 17-10** The Macintosh Portable 68000 Direct Slot expansion card



Physical guidelines for Macintosh LC expansion cards

This section provides the physical information you need to design an expansion card for the Macintosh LC computer. The information includes mechanical drawings showing dimensions and component mounting restrictions.

Figure 17-11 gives the maximum length and width of the Macintosh LC expansion card and shows the location of the 96-pin connector. Figure 17-12 provides component location and height restrictions for the Macintosh LC expansion card. The Macintosh LC uses the same Euro-DIN 96-pin expansion connector described earlier in this chapter in the section "The 68000 Direct Slot 96-Pin Connector for the Macintosh SE." The connectors for the expansion card and the main logic board are the same as those shown for the Macintosh SE in Figures 17-4 and 17-6.

▲ **Warning** The component locations and height restrictions shown in Figure 17-12 are critical to your expansion card design. Failure to adhere to these specifications could cause card failure and possible damage to the main logic board. ▲

Figure 17-13 is a design guide for the shield plate that is required with the Macintosh LC expansion card to maintain EMI/RFI (electromagnetic interference/radio-frequency interference) integrity. Figure 17-14 shows the steps for positioning the expansion card on the main logic board. Figure 17-15 shows the dimensions of the plastic supports used for expansion cards in the Macintosh LC.

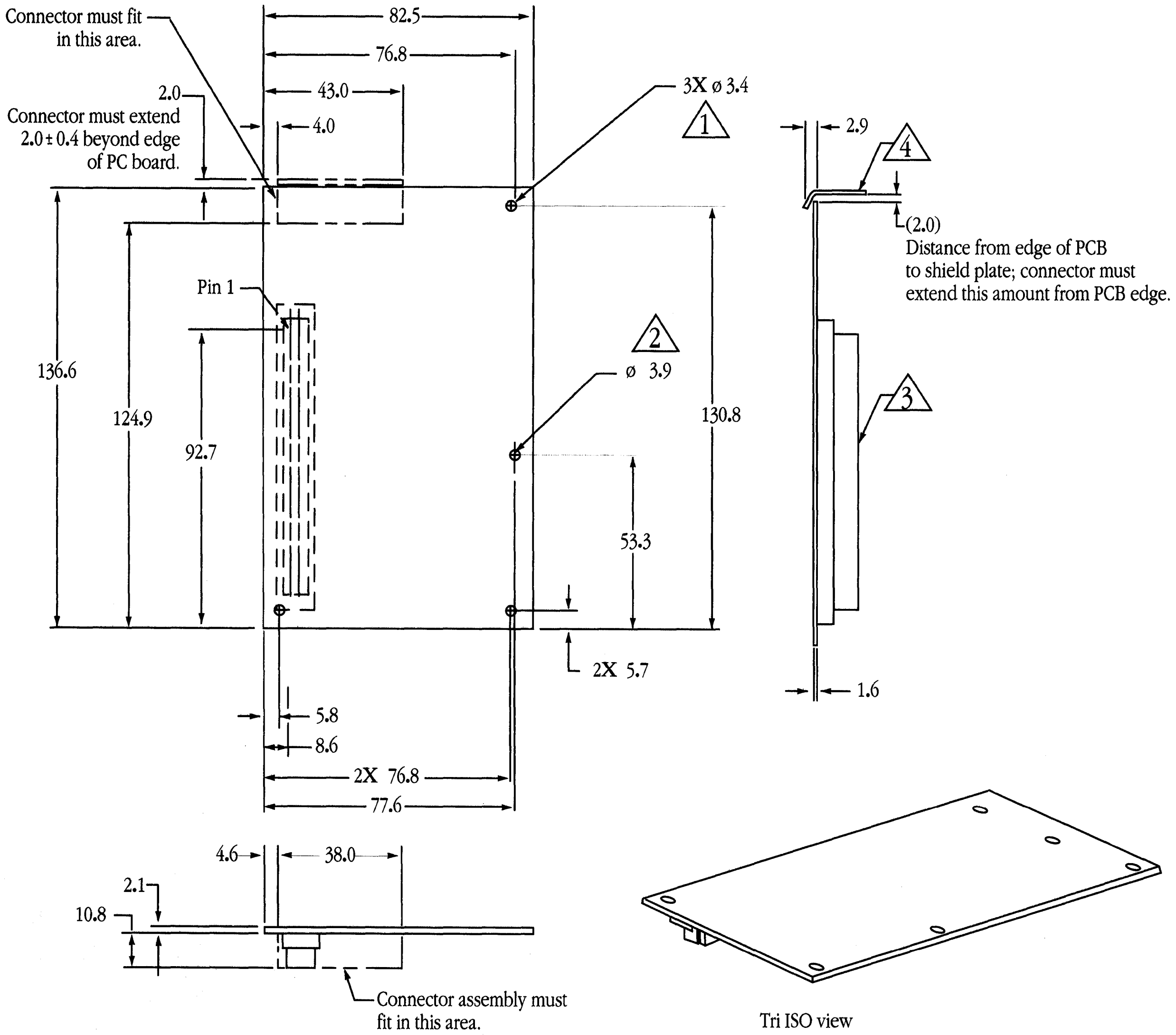
■ **Figure 17-11** Macintosh LC expansion card design guide

△1 Tooling holes; used for standoff.

△2 Hole for standoff.

△3 96-pin connector.

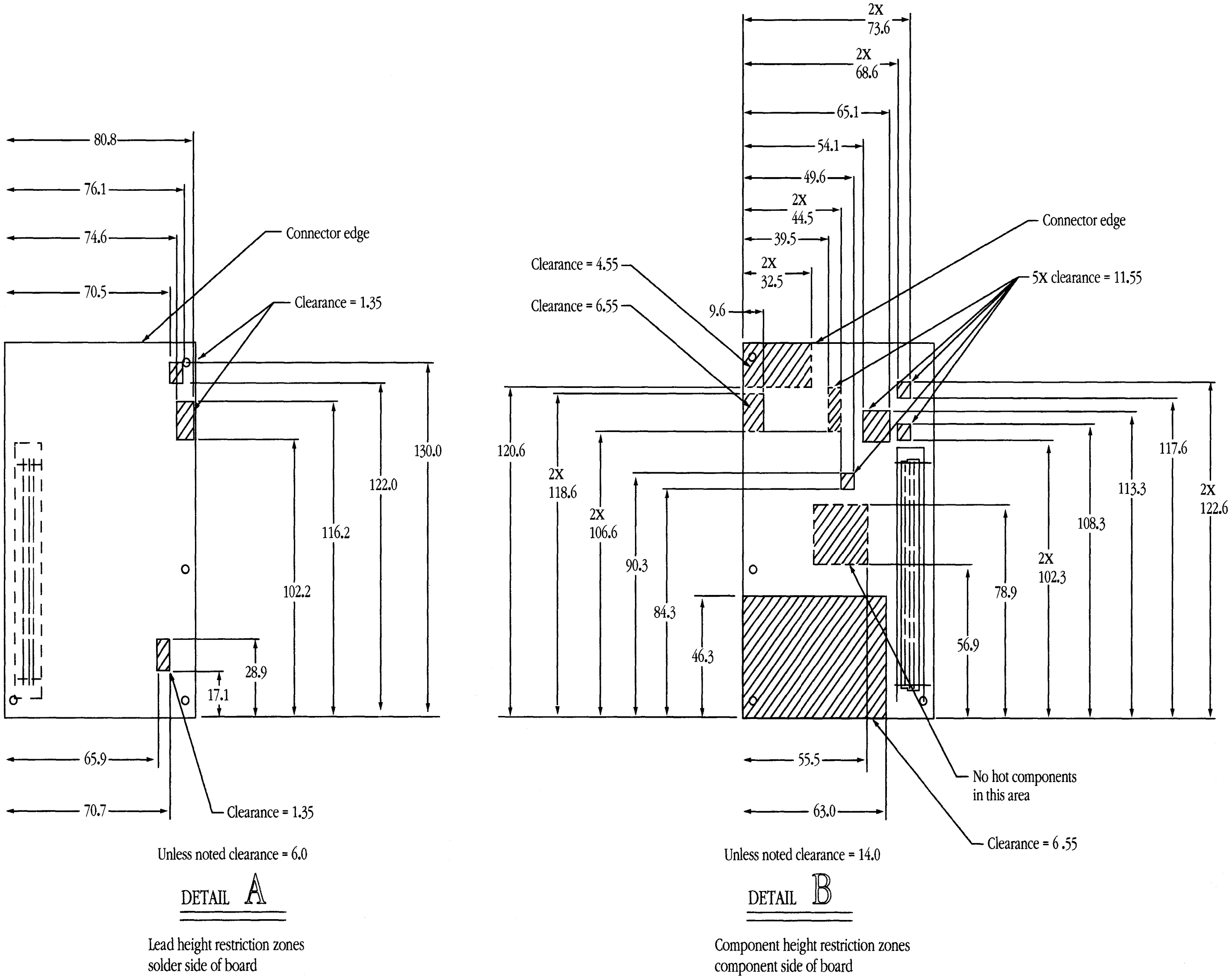
△4 Shield plate required to maintain integrity of EMI/RFI seam.



Connector must fit in this area.
2.0
Connector must extend 2.0 ± 0.4 beyond edge of PC board.
82.5
76.8
43.0
4.0
3X ø 3.4
1
2.9
4
(2.0)
Distance from edge of PCB to shield plate; connector must extend this amount from PCB edge.
Pin 1
136.6
124.9
92.7
2
ø 3.9
130.8
3
53.3
2X 5.7
1.6
5.8
8.6
2X 76.8
77.6
4.6
38.0
2.1
10.8
Connector assembly must fit in this area.
Tri ISO view

Dimensions are in millimeters.

Figure 17-12 Macintosh LC expansion card component location and height restrictions

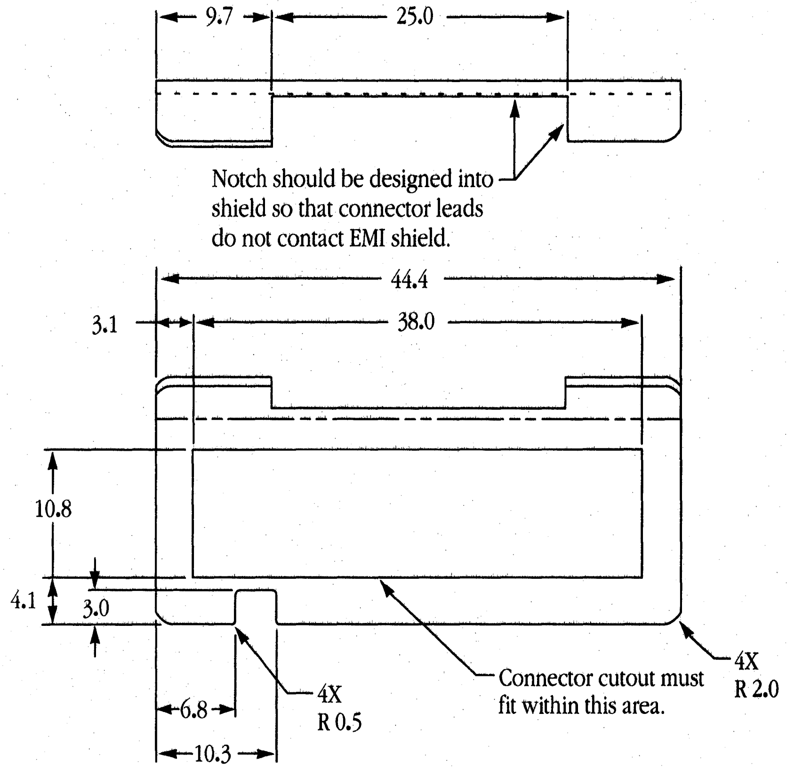


Dimensions are in millimeters.

■ **Figure 17-13** Design guide for Macintosh LC expansion card shield plate

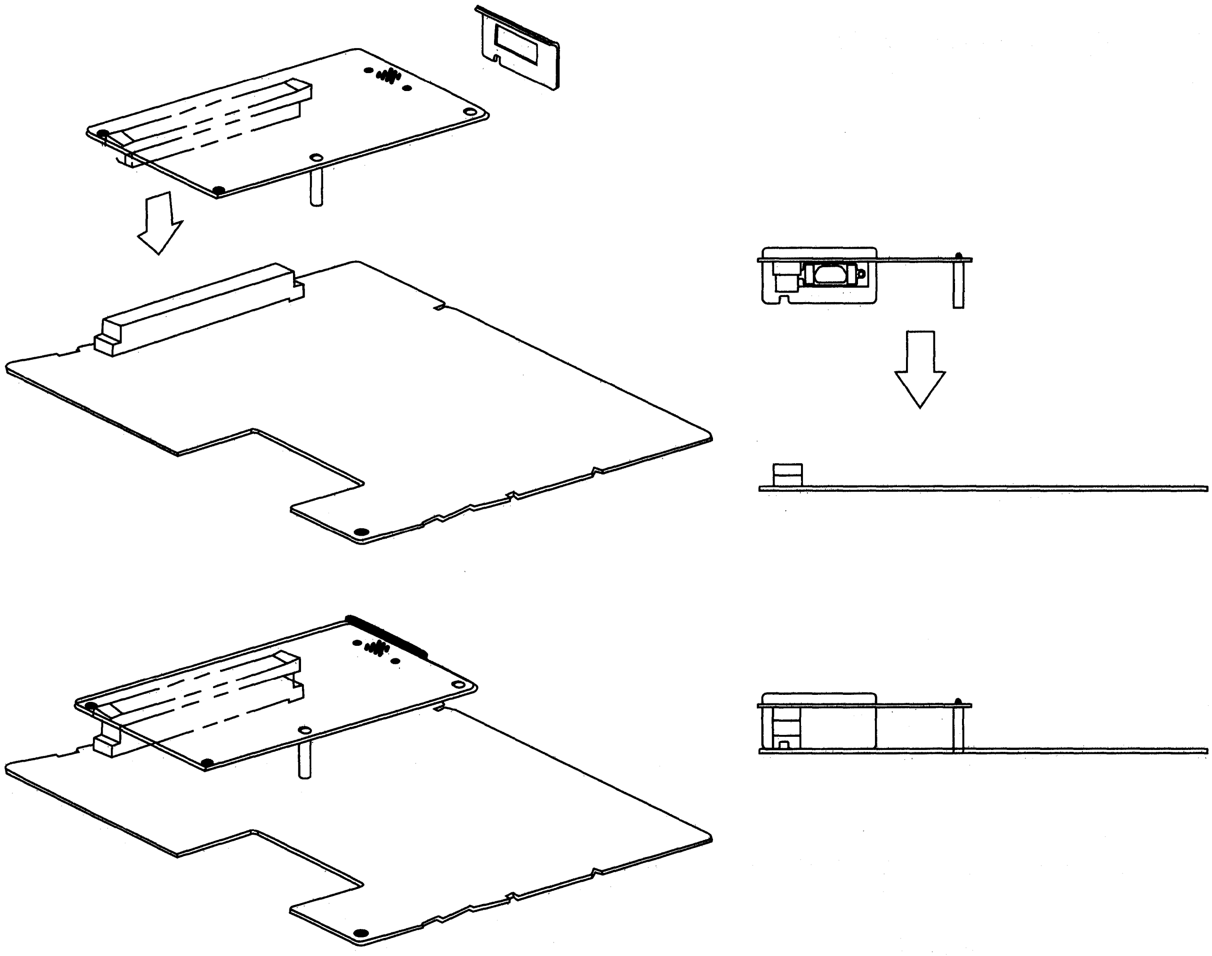
NOTES: Unless otherwise specified

1. Interpret all dimensions and tolerances per ANSI Y14.5-1982.
2. Material: 1.00 mm thick, AISI type 1020, cold rolled steel.
   Finish: aluminum-silicon alloy coating,
   T125-T14 or Apple product design approved equivalent.
3. Remove all burrs and sharp edges.

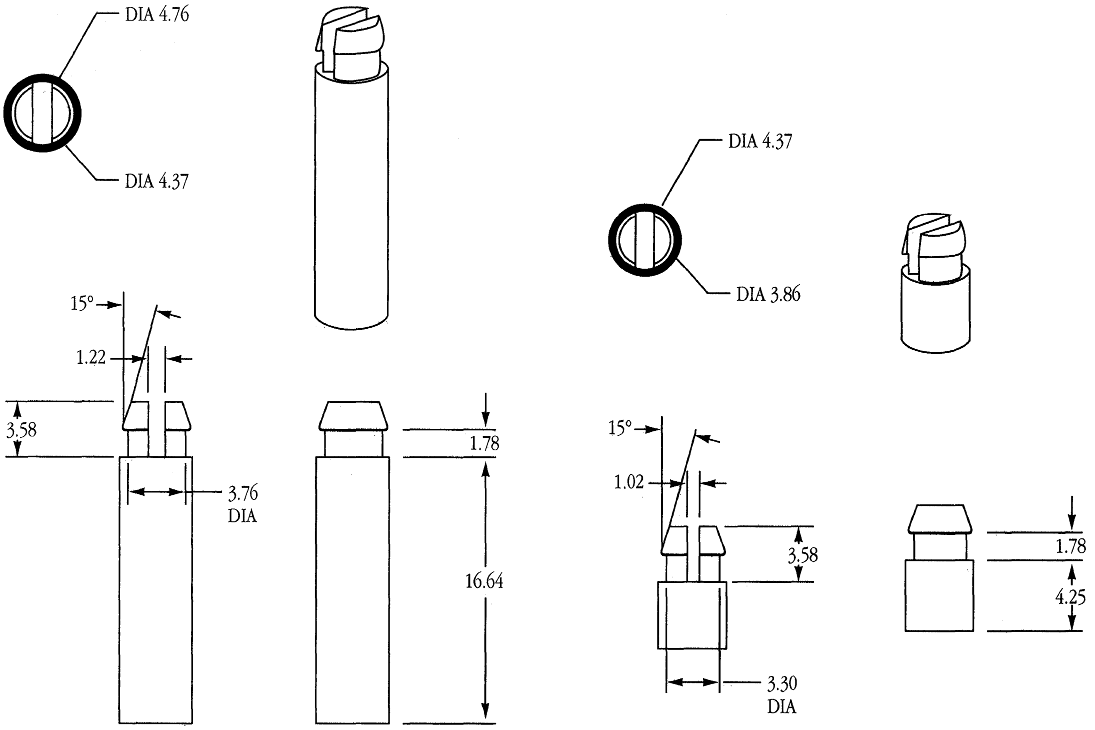

9.7
25.0
Notch should be designed into shield so that connector leads do not contact EMI shield.
44.4
3.1
38.0
10.8
4.1
3.0
6.8
10.3
4X
R 0.5
Connector cutout must fit within this area.
4X
R 2.0

Dimensions are in millimeters.

- **Figure 17-14** Positioning the expansion card on the Macintosh LC main logic board



■ **Figure 17-15** Plastic supports for Macintosh LC expansion cards



16.55 mm

1. Material: 6/6 nylon or Apple engineering approved equivalent.
2. Standoff for use in 3.86 diameter hole and 1.60 thick panel.

4.25 mm

1. Material: 6/6 nylon or Apple engineering approved equivalent.
2. Standoff for use in 3.4 diameter hole and 1.60 thick panel.

Dimensions are in millimeters.

Macintosh LC external access opening

An opening in the rear of the Macintosh LC case allows an expansion card to communicate with external devices. This opening accommodates a DB-15 connector, which you can include as an integral part of your expansion card design. The connector attaches to the expansion card in the area shown in Figure 17-11.

Expansion card installation for the Macintosh LC

The expansion card mounts parallel to the Macintosh LC main logic board, with component sides facing each other (see Figure 17-14). It is important that you adhere to height restrictions and do not place hot components in those areas called out in Figure 17-12. You maintain the EMI/RFI integrity of the system by installing the shield plate (Figure 17-13) between the card and the external access opening on the rear of the Macintosh LC case.

Physical guidelines for Macintosh SE/30 expansion cards

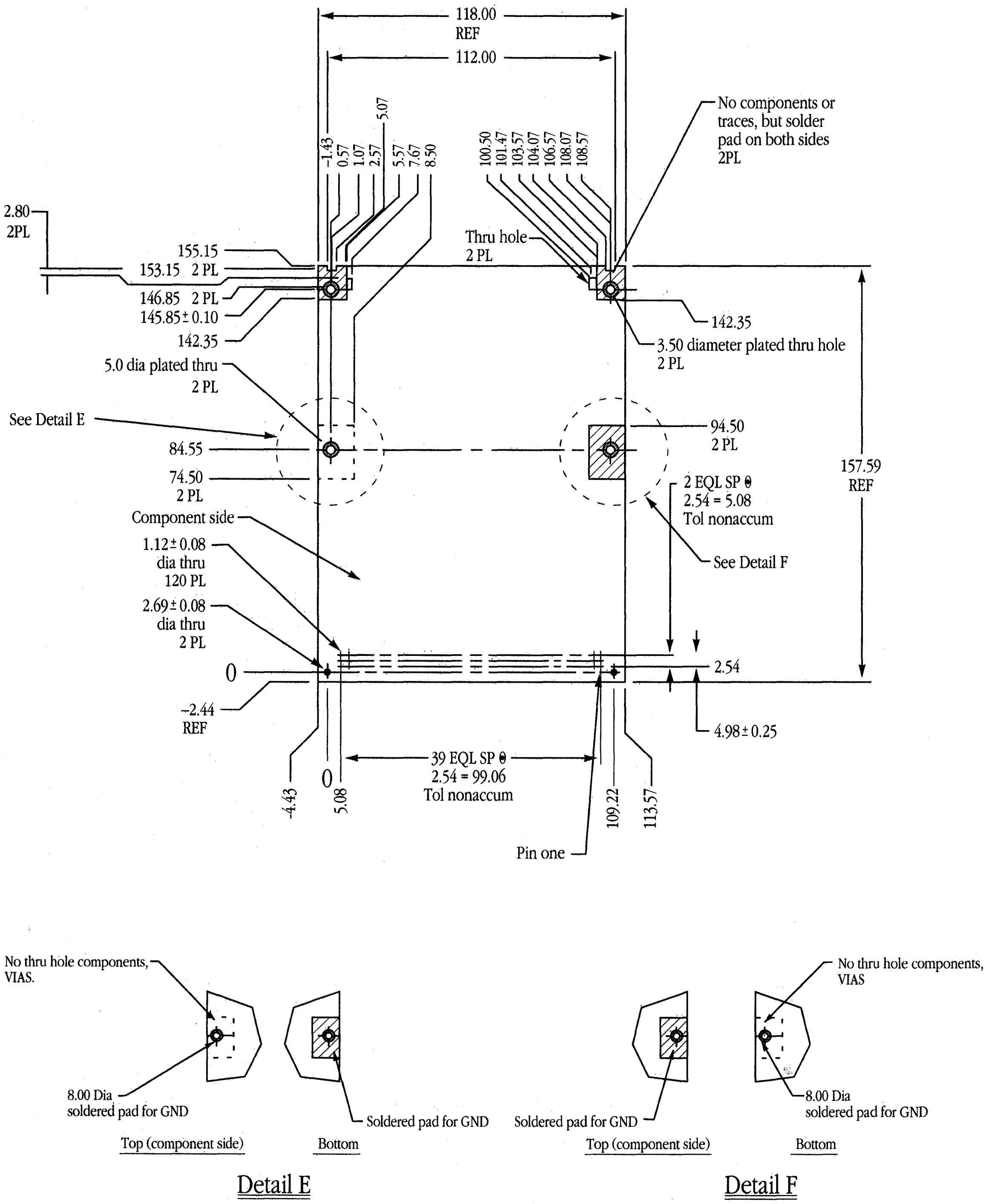
This section provides mechanical drawings that show the spatial relationship between an expansion card and the Macintosh SE/30 main logic board.

Figures 17-16 through 17-18 show design considerations for expansion cards that can be used in the Macintosh SE/30 and possibly in future 68030-based machines. Notice that you can design your card in either of two different sizes: Figure 17-16 shows the smallest allowable card size, and Figure 17-17 shows the largest allowable card size. Figure 17-18 shows the largest component heights allowed on the two different card sizes.

Figure 17-19 is a design guide for the Macintosh SE/30 main logic board. You should pay particular attention to the design of the main logic board and the Macintosh SE/30 chassis to make sure that components on your expansion card do not interfere with mounting hardware.

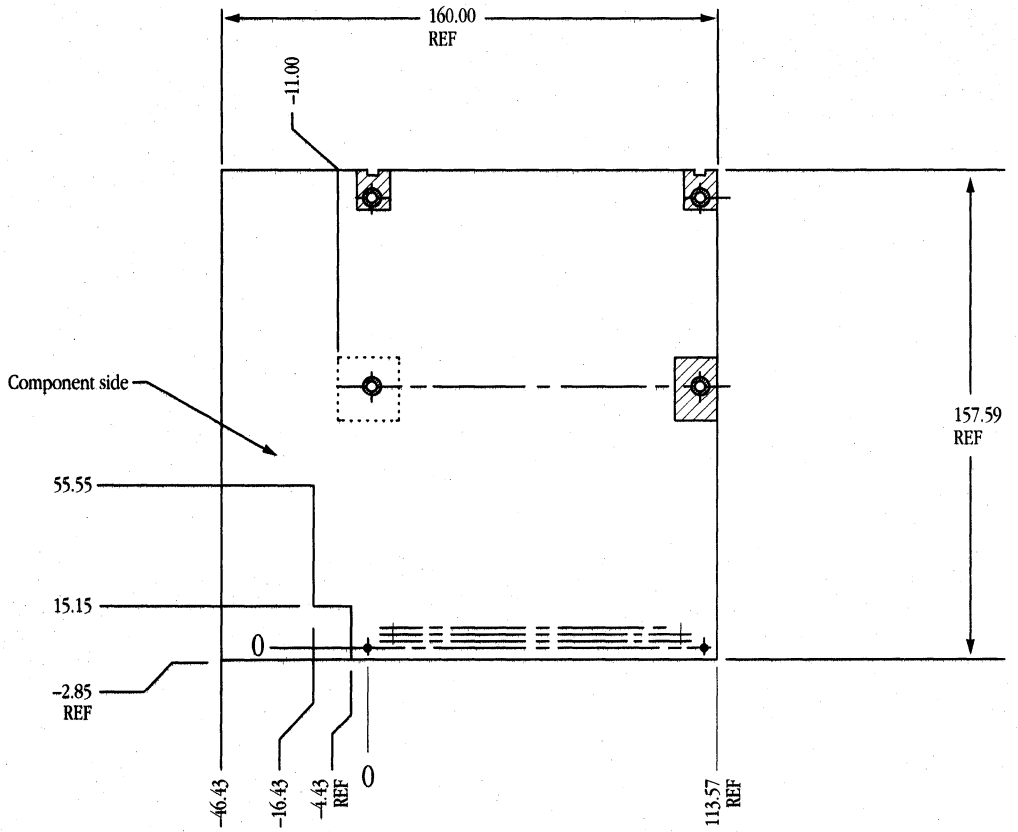
Figure 17-20 shows how an expansion card mounts in the Macintosh SE/30 chassis. Figure 17-21 shows how the expansion card mounting clips should be oriented for two different revision levels of the main chassis.

■ **Figure 17-16** Smallest allowable Macintosh SE/30 expansion card



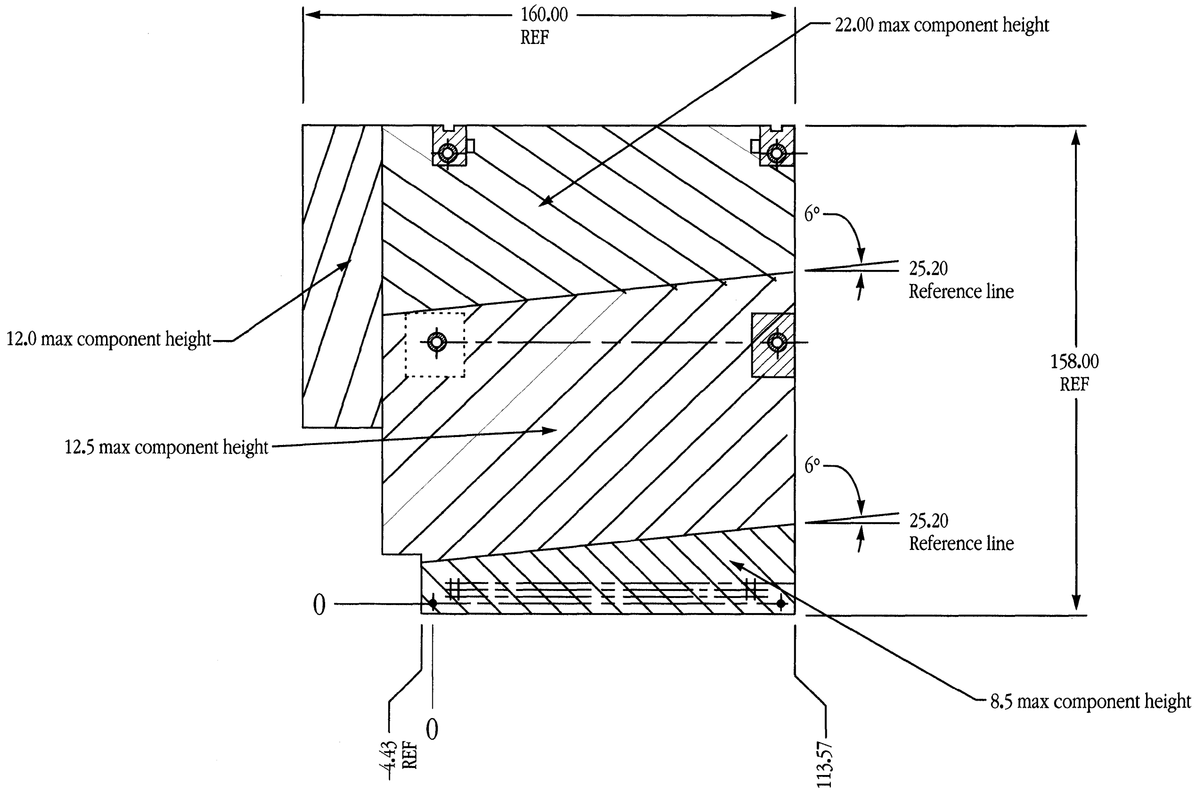
Dimensions are in millimeters.

■ **Figure 17-17** Largest allowable Macintosh SE/30 expansion card



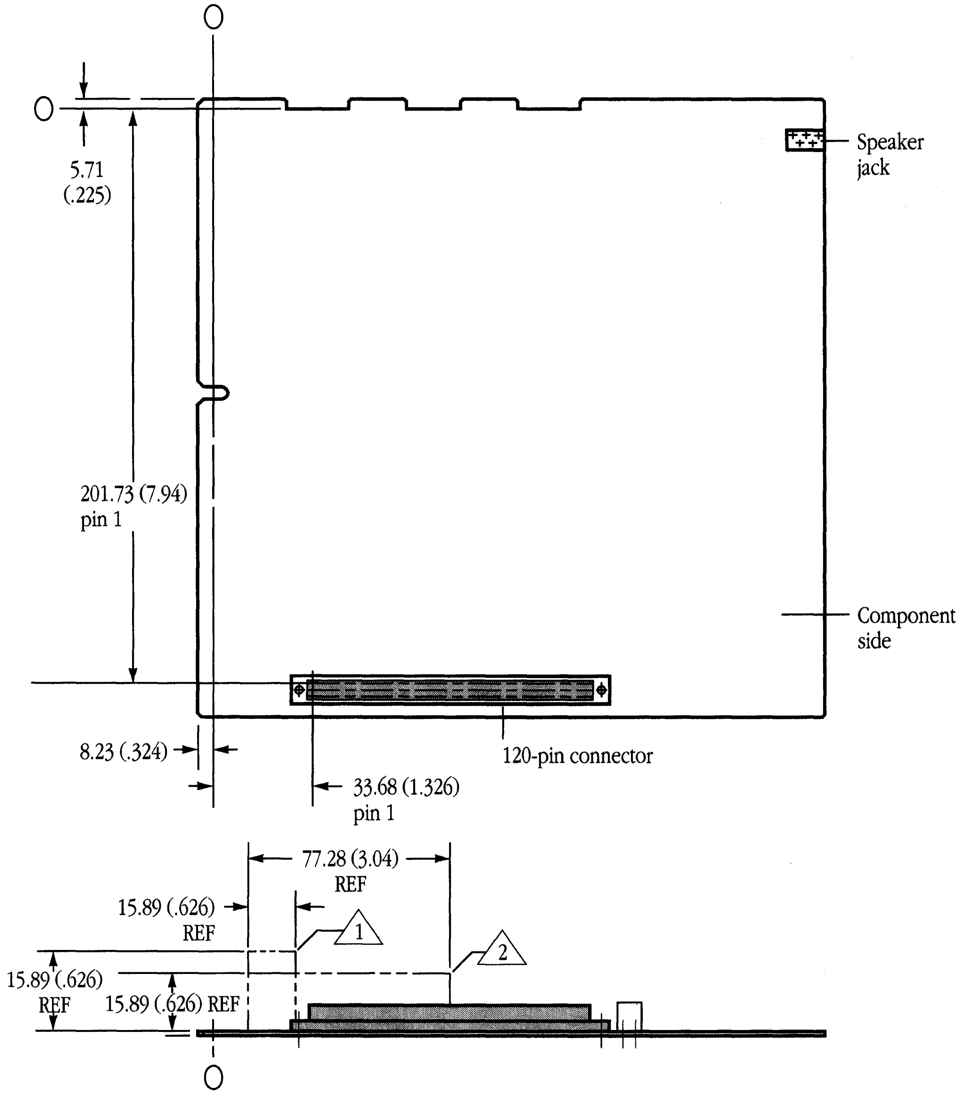
Dimensions are in millimeters.

■ **Figure 17-18** Largest allowable component heights for a Macintosh SE/30 expansion card



Dimensions are in millimeters.

■ **Figure 17-19** Expansion connector on the Macintosh SE/30 main logic board

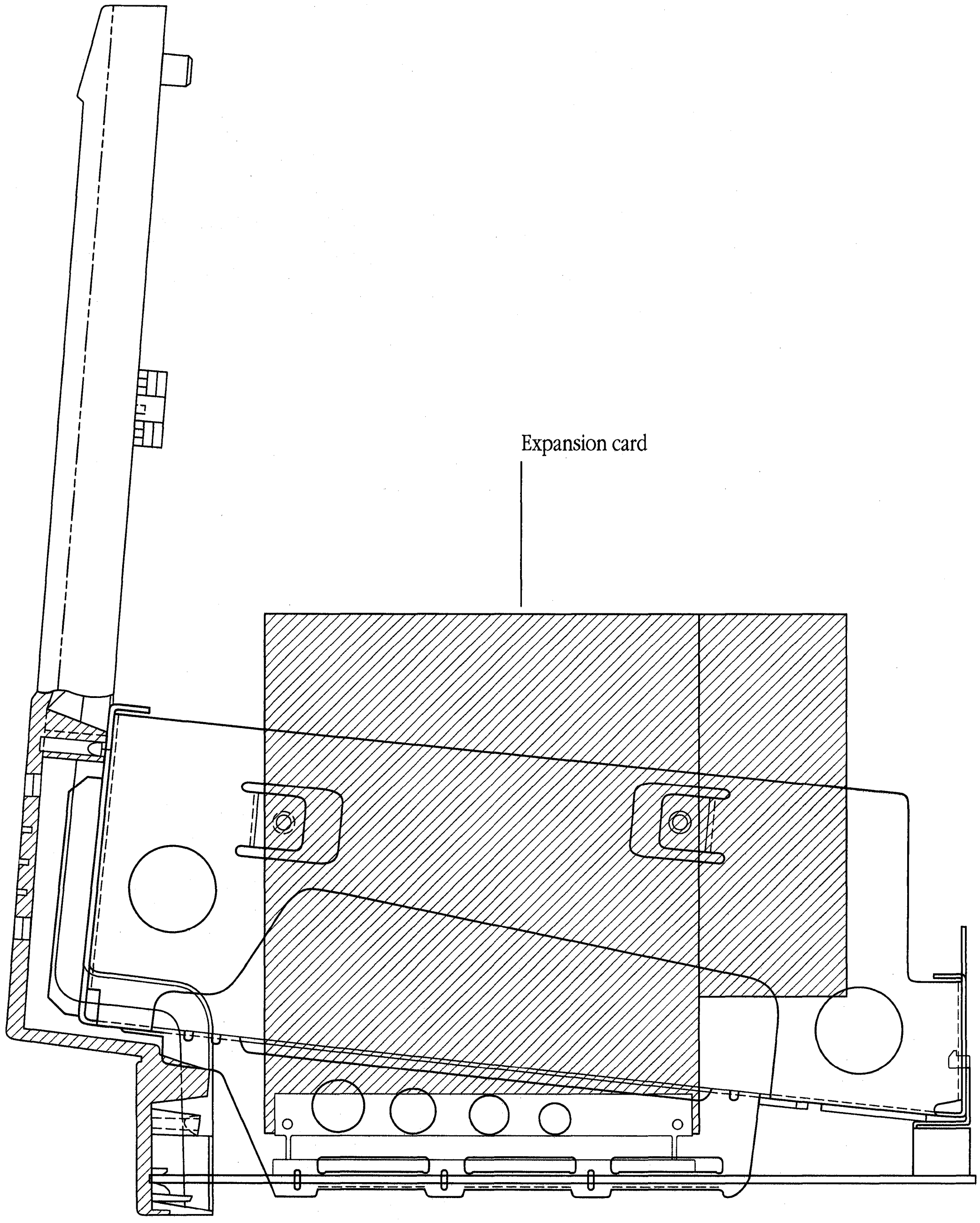


1 Indicated area represents space available for 31.50 (1.24) high (including socket) SIMMs module.

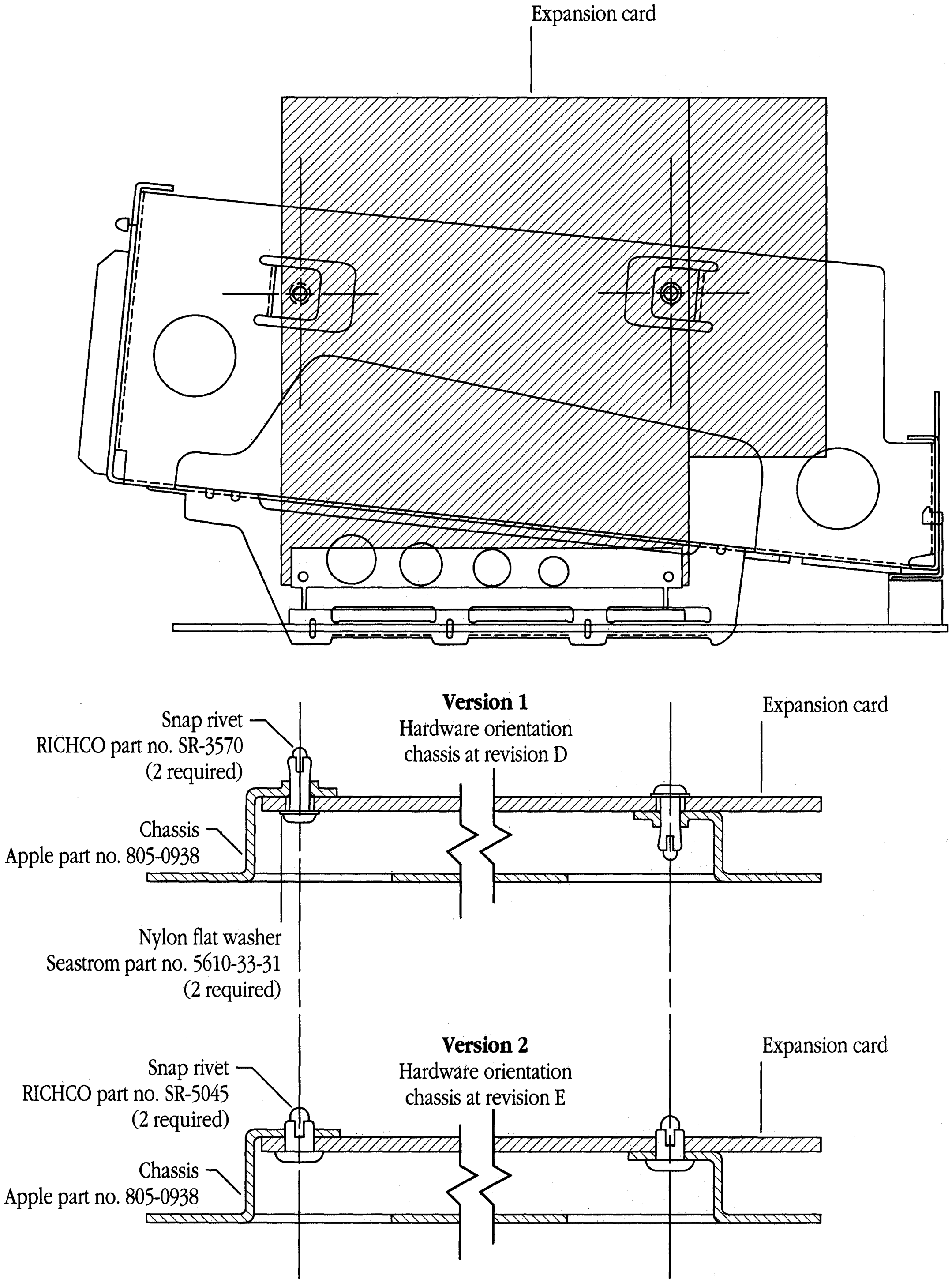
2 Indicated area represents space available for 24.00 (.94) high (including socket) SIMMs module.

Dimensions are in millimeters with inches in parentheses.

- **Figure 17-20** An expansion card in the Macintosh SE/30 assembly



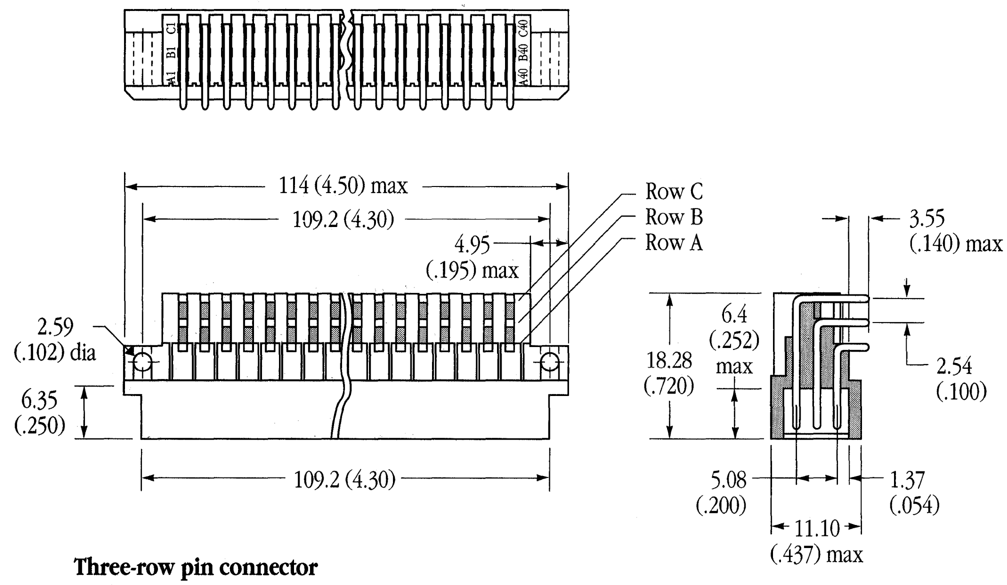
■ **Figure 17-21** Orientation of Macintosh SE/30 mounting hardware



The 68030 Direct Slot 120-pin connector for the Macintosh SE/30

Figure 17-22 shows the plug connector that mates with the Euro-DIN 120-pin socket connector on the main logic board. Figure 17-19 shows the location of the 120-pin socket connector on the main logic board assembly. Figure 17-23 gives the prominent details of the socket connector.

- **Figure 17-22** A 120-pin plug connector for a Macintosh SE/30 expansion card

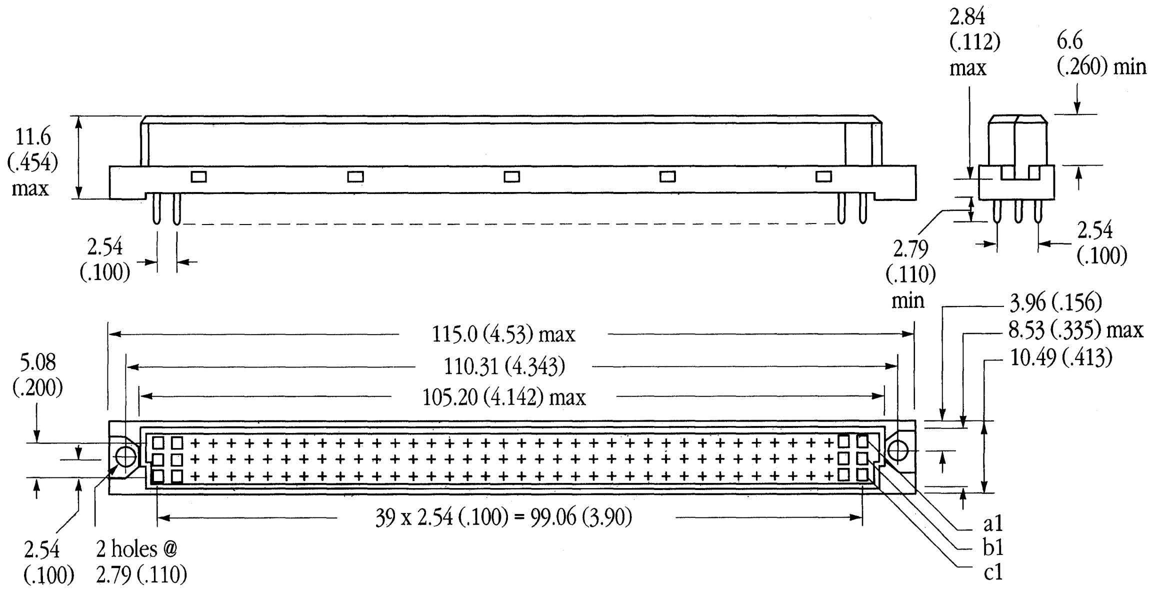


Three-row pin connector

120 contact positions
2.54 mm (.100 inch) spacing pins
Gold plated, 20 microinches, over nickel plate

Dimensions are in millimeters with inches in parentheses.

■ **Figure 17-23** Detail of 120-pin socket connector used on Macintosh SE/30 main logic board



Three-row socket connector

120 contact positions
2.54 mm (.100 inch) spacing sockets
Gold plated, 20 microinches, over nickel plate

Dimensions are in millimeters with inches in parentheses.

External connection for a Macintosh SE/30 expansion card

This section discusses both electrical and physical considerations required in making connections to external equipment for the Macintosh SE/30.

The Macintosh SE/30 computer has an external device access opening through which another piece of equipment can be connected. Typically, a cable would be routed from the expansion card, upward through a cutout in the back of the chassis, and then to a connector on a connector card you provide.

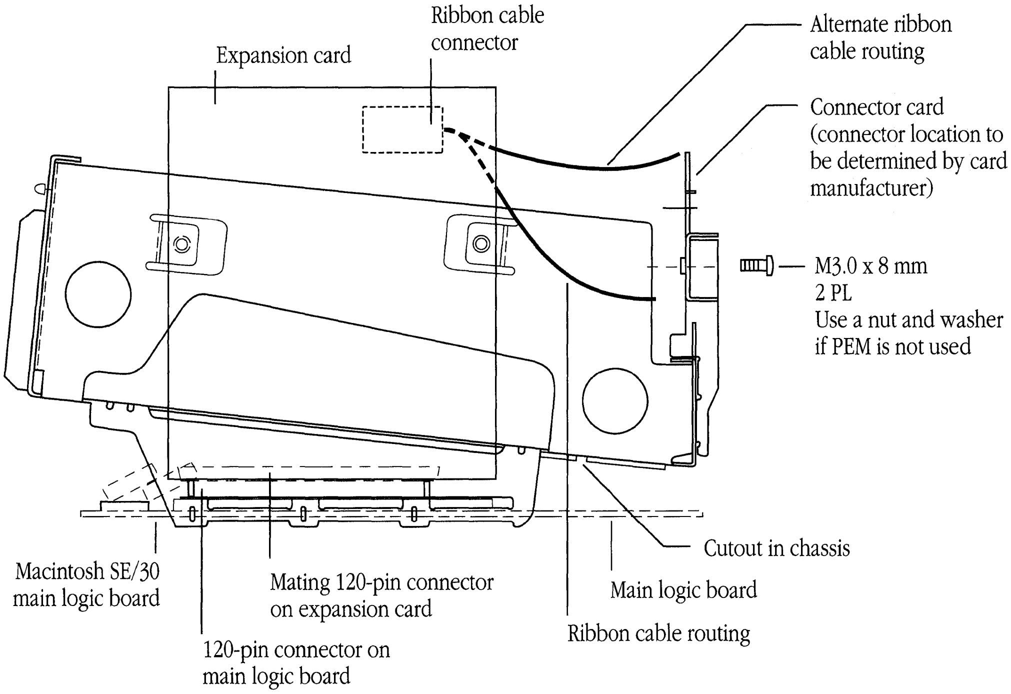
The drawing in Figure 17-7 shows the sheet-metal and structural requirements for mounting a connector card in a Macintosh SE. Figure 17-7 also applies to a Macintosh SE/30 except that the expansion card on a Macintosh SE/30 is mounted vertically, not horizontally, as in the Macintosh SE.

Figure 17-24 shows the recommended cable-routing paths for the Macintosh SE/30. Notice that the minimum allowable length for the internal cable on a Macintosh SE/30 is 220 mm (8.6 inches). If you adhere to this cable length, your internal cable can also be used to connect expansion cards to a Macintosh SE, a Macintosh IIsi, and other Macintosh computers.

Foldout 8 at the end of the book is a design guide for the connector card. All areas of significant importance are noted on the drawing. If you design a connector card that adheres to the dimensions in Foldout 8, it can be used on the Macintosh SE, the Macintosh SE/30, and future versions of the compact Macintosh.

▲ **Warning** Foldout 8 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change. ▲

■ **Figure 17-24** Internal expansion cable routing for Macintosh SE/30


Ribbon cable connector
Expansion card
Alternate ribbon cable routing
Connector card (connector location to be determined by card manufacturer)
M3.0 x 8 mm
2 PL
Use a nut and washer if PEM is not used
Cutout in chassis
Macintosh SE/30 main logic board
Mating 120-pin connector on expansion card
Main logic board
Ribbon cable routing
120-pin connector on main logic board

Note: The length of the ribbon cable must be 220 millimeters (8.6 inches) or longer between connectors.

Physical guidelines for Macintosh IIfx PDS expansion cards

The expansion card for the 68030 Direct Slot of a Macintosh IIfx computer is identical in size to a NuBus card but uses a 120-pin plug connector instead of the 96-pin plug connector used on a NuBus card. When installed in the computer, the PDS expansion card takes the place of a NuBus card in slot $E and prevents it from using that address space. The Macintosh IIfx can accommodate a maximum of either six NuBus cards, or five NuBus cards and one PDS card.

Foldout 9 at the back of the book shows the pertinent physical details you need to design a PDS expansion card for the Macintosh IIfx computer. This drawing shows the overall dimensions and the connector placement and provides clearance dimensions for installing the card in the Macintosh IIfx computer.

▲ **Warning** Foldout 9 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change. ▲

The 120-pin plug connector used on a Macintosh IIfx PDS expansion card is physically identical to the connector shown for the Macintosh SE/30 PDS expansion card in Figure 17-22. The 120-pin socket connector used on the main logic board of the Macintosh IIfx computer is physically identical to the connector shown for the main logic board of the Macintosh SE/30 computer in Figure 17-23.

Physical guidelines for Macintosh IIsi Direct Slot expansion cards

To install a 68030 Direct Slot expansion card in the Macintosh IIsi, you must first install a Direct Slot adapter card. A 68030 Direct Slot adapter kit is available from an authorized Apple dealer. The following sections describe the installation of the adapter card and the Macintosh IIsi Direct Slot expansion cards.

Physical implementation of the Macintosh IIsi 68030 Direct Slot adapter kit

The 68030 Direct Slot adapter card for the Macintosh IIsi includes two 120-pin connectors. One is a plug connector that mates with the Euro-DIN 120-pin socket connector located on the left side (looking from the front) of the computer's main logic board. The adapter card mounts vertically in this connector. The other connector on the adapter card is a 120-pin socket connector (the same as the main logic board connector) into which the expansion card is installed. The expansion card is mounted horizontally over the main logic board. Figure 17-25 is a sketch showing a processor-direct slot expansion card and its adapter card installed on the main logic board of a Macintosh IIsi computer.

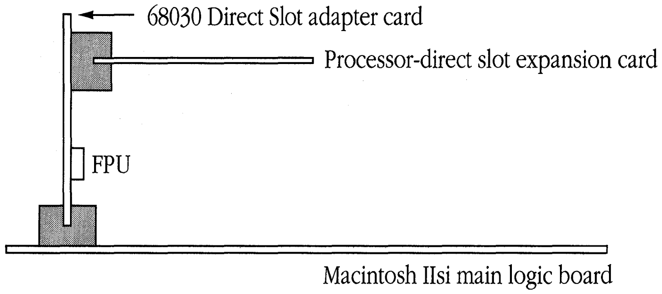
The 68030 Direct Slot adapter kit includes a snap-in bracket that mounts to the power supply frame to support the expansion card and secure it to the machine.

You can get the snap-in bracket that meets Apple specifications from

SPM, Inc.
Anaheim, CA 92806

The part number for the snap-in bracket is 815-6246. Before ordering the snap-in bracket, however, you must first obtain authorization from Macintosh Developer Technical Support (MacDTS).

■ **Figure 17-25** Installing a PDS card and adapter on the Macintosh IIsi main logic board


68030 Direct Slot adapter card
Processor-direct slot expansion card
FPU
Macintosh IIsi main logic board

External connections for the Macintosh IIsi PDS expansion card

The design of the Macintosh IIsi computer allows the use of existing hardware to connect an expansion card to the external world. This hardware is supplied with the expansion card and includes a small connector card, two screws, and a ribbon cable. You install the connector card (which is the same size as that specified for a Macintosh SE or a Macintosh SE/30 computer) to the opening in the rear of the Macintosh IIsi computer. You then plug the ribbon cable into a connector on the expansion card and route it underneath the expansion card to the connector card installed on the rear of the Macintosh IIsi computer. Current Macintosh SE/30 cards will work if their ribbon cables are at least 220 mm (8.6 inches) long between connectors.

◆ *Note:* The specifications for the connector card and ribbon cable have been chosen to ensure compatibility with future Macintosh computers. Detailed information on connector card specifications is provided in Foldout 8, "Connector Card Design Guide for Macintosh PDS Computers."

To mount a connector card in the Macintosh IIsi, you must include appropriate mounting holes on your connector card as well as two screws for attaching the card to the chassis. The required screw size is identical to the screw size for the Macintosh SE and is shown in Figure 17-7.

Design considerations for Macintosh IIsi PDS expansion cards

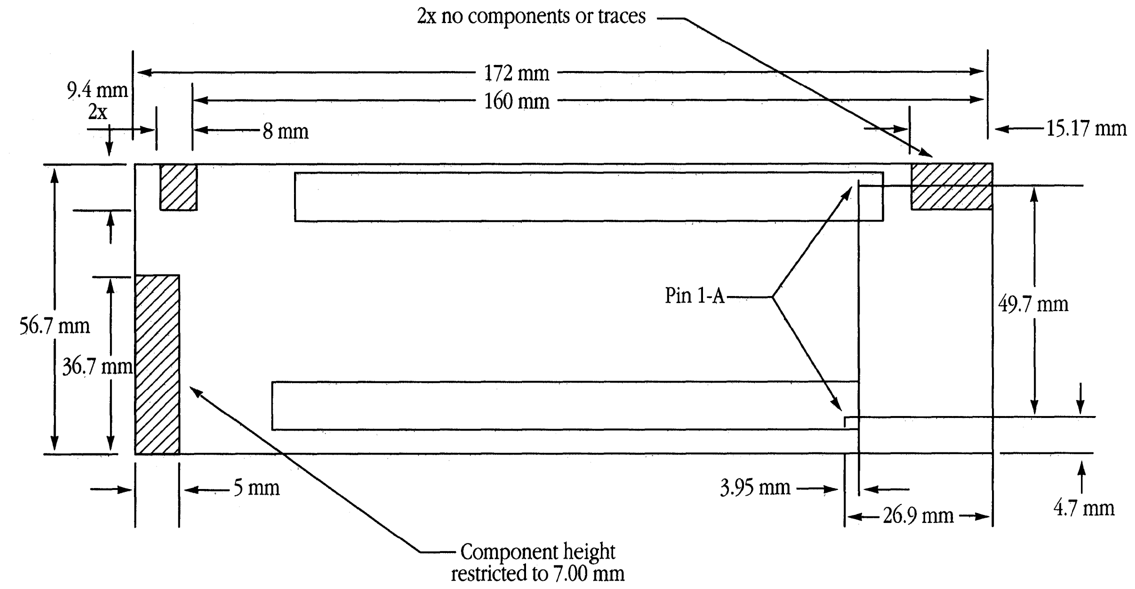
Some important factors should be considered when using existing PDS expansion cards or designing new cards for the Macintosh IIsi computer.

- The fact that the ribbon cable from the expansion card to the connector card must travel the entire length of the main logic board could have EMI implications for third-party cards. Therefore, at the very least, cards should be tested to make sure that they still comply with FCC guidelines. In some cases, cables may need to be shielded or include ferrite sleeves to avoid violating the FCC specifications. At worst, the layout or design of an expansion card may need to be modified.
- Apple strongly recommends against designing cards to fit directly into the 120-pin connector on the main logic board (without an adapter card) because the vertical position of the expansion card may cause EMI or thermal problems.
- Any PDS expansion cards designed to fit in a Macintosh SE/30 computer can be used in the Macintosh IIsi computer. Figures 17-16 through 17-18 are design guides for these expansion cards.

Macintosh IIsi adapter cards

Even though Apple provides an MC68030 Direct Slot adapter kit, you may want to develop your own adapter card for the Macintosh IIsi. The electrical specifications for the adapter card are included in Chapter 15. Figure 17-26 shows the outline of the MC68030 PDS adapter card.

■ **Figure 17-26** Macintosh IIsi PDS adapter card outline



Physical guidelines for 68040 Direct Slot expansion cards

A 68040 Direct Slot expansion card for the Macintosh Quadra 700 and Macintosh Quadra 900 computers has the same dimensions as a NuBus card and can include a back-panel connector. Foldout 2 provides information about the size of the expansion card and the location of the 140-pin PDS connector. Foldout 2 also shows a NuBus connector; however, you should omit this connector when designing a PDS expansion card for the Macintosh Quadra 700 or Macintosh Quadra 900. Foldout 3 provides I/O clearance dimensions for a NuBus card or a PDS card being installed in a Macintosh Quadra 900. Foldout 4 provides similar requirements for a NuBus or a PDS card being installed in a Macintosh Quadra 700. Refer to Chapter 6, "NuBus Card Physical Design Guide," for more information about the physical specifications for NuBus cards in the Macintosh Quadra 700 and the Macintosh Quadra 900.

Because NuBus cards for the Macintosh Quadra 900 can be oversized, 68040 Direct Slot expansion cards can be oversized as well. Refer to Foldout 5 for the specifications for the oversized Macintosh Quadra 900 NuBus and processor-direct expansion cards.

PDS expansion cards for the Macintosh Quadra–family computers must have a 140-pin plug connector to fit in the 140-pin socket connector on the main logic board. The plug connector should have compliant pins (force-fit insertion) rather than solder-type pins for connection to the expansion card if components are to be mounted on the top side of the card.

You can order 140-pin connectors meeting Apple specifications from

KEL Connectors, Incorporated
Sunnyvale, CA 94086

The connector on the Macintosh Quadra 700 and Macintosh Quadra 900 main circuit boards is KEL part number 8817-140-170SH. The corresponding connector on the 68040 Direct Slot expansion card is KEL part number 8807-140-170LH.

Chapter 18 **Processor-Direct Slot Design Example**

This chapter contains a performance-proven example of processor-direct slot expansion card design. It describes the electrical and interface characteristics of a simple disk controller card that allows the Macintosh SE processor to communicate with a generic disk drive through the 68000 Direct Slot.

Disk controller overview

The disk controller card allows for one drive to be connected to the Macintosh SE through the cable supplied. The disk controller is inexpensive, but is capable of two software-selectable recording formats: **frequency modulation (FM)** and **modified frequency modulation (MFM)**. FM is an IBM 3740–compatible, single-density format. MFM is an IBM System 34–compatible, double-density format.

The disk controller card plugs into the 96-pin expansion connector on the main logic board of the Macintosh SE and connects to a floppy disk drive located outside the Macintosh SE. The installation of this card and its associated cables is intended to be done by dealers and not by end users. The disk controller card consists of a disk controller IC and a disk interface IC, a DMA controller IC, some buffers, and three PALs. All controlling firmware and sector-buffering RAM exist in the Macintosh SE.

The control registers are mapped into the address space of the Macintosh SE from $80 0000 through $8F FFFF. No other address space is memory mapped to the controller.

System configuration

The controller package inside the Macintosh SE consists of a disk controller expansion card, a 26-wire flat ribbon cable, and a connector card.

The disk controller card connects to the Macintosh SE processor through the 96-pin expansion connector on the main logic board assembly. A 6-inch-long ribbon cable ties the disk controller card to the connector card.

The connector card, which mounts to the bracket behind the external device access opening, has two connectors. One connector is a 26-pin connector, which terminates the 6-inch ribbon cable from the internal controller card. The other connector is a DB-37 into which the external disk drive can be plugged via the cable supplied with that drive. See Figure 17-7, Figure 17-8, and Foldout 8 for drawings depicting the configuration.

Interface card block diagram

Figure 18-1 is a block diagram of the floppy disk controller. The controller card is made up of the following parts:

Control PALs: The control PALs provide the address decoding and timing control for the disk controller. They memory map the various control and status registers of the disk controller into the Macintosh SE address space $80 0000 through $8F FFFF.

Data bus transceivers: The data transceivers provide multiplexing control and sufficient current drive to and from the controller onto the data bus. During high-byte transfers data is placed on lines D8–D15, while during low-byte transfers the data goes on lines D0–D7.

Status driver: The status driver allows three signals to be read by the Macintosh SE: disk drive selected, disk controller interrupt (INT), and disk change.

Disk controller IC: The disk controller IC contains the circuitry necessary to connect to the generic disk drive. Coupled with the companion disk interface IC, this LSI chip handles all operations with the drive, including reading and writing data, formatting, seeking, sensing drive status, and recalibrating.

Disk interface IC: The disk interface IC provides drive and timing support to the disk controller IC. It contains write precompensation and phase-locked loop circuitry.

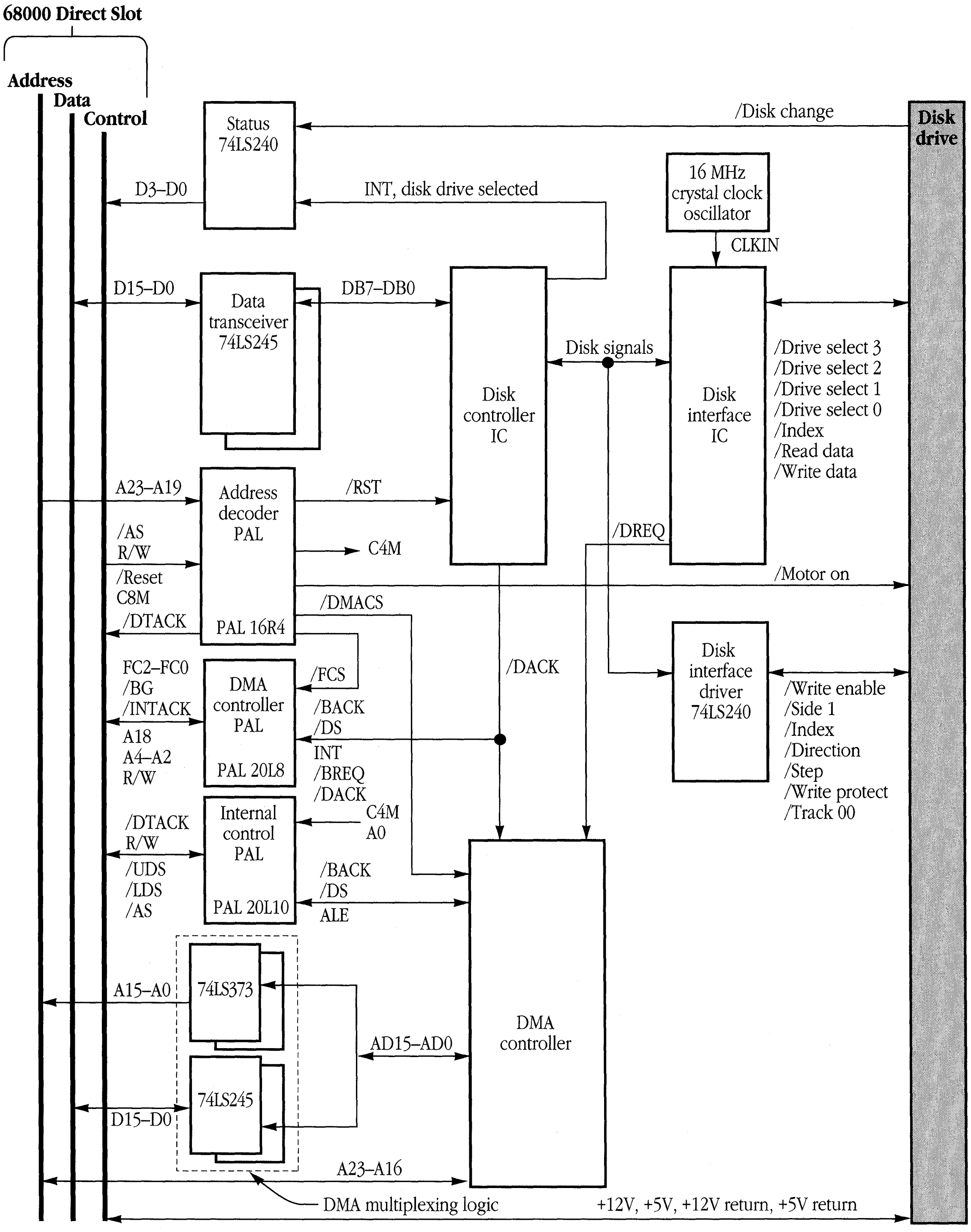
Disk interface driver: The disk interface driver buffers and provides current drive for several signals coming from and going to the disk drive. It also is used as a multiplexer for four of these signals.

16 MHz crystal clock oscillator: The 16 MHz crystal clock oscillator provides a 16 MHz clock to the disk interface IC for use in the drive interface.

Dual-channel DMA controller PAL: The DMA controller handles all DMA data-transfer operations between the disk controller IC and the Macintosh SE memory.

DMA address and data multiplexing logic: The dual-channel DMA controller has a multiplexed address and data bus. The multiplexing logic is used to demultiplex this bus. The logic consists of two 74LS373's and two 74LS245's.

■ **Figure 18-1** Floppy disk controller block diagram



Floppy disk controller logic

The disk drive control is provided by the disk controller IC, disk interface IC, and some 74LS240 drivers. The disk controller IC is the controlling chip and communicates with the disk interface IC. Details of this logic are not directly relevant to the 68000 Direct Slot interface and so are not given here.

Macintosh SE interface logic

The controller communicates with the 68000 Direct Slot via several drivers and PALs. The controller follows the timing of the MC68000 processor whether in PIO (programmed input/output) or DMA (direct memory access) transfers. Certain key signals are described in Table 18-1.

■ **Table 18-1** Bus control signals

Signal name	Signal description
ALE	Signals that the DMA controller is gating a valid address onto the multiplexed address/data lines AD0–AD15
/AS	Indicates that a valid address is on the address bus
/BACK	Indicates that the DMA controller has the processor bus
/BG	Signals to the controller that it owns the processor bus after completion of the current bus cycle
/BR	Signals that the controller card would like to own the processor bus in order to perform a DMA transfer
/BREQ	Indicates that the DMA controller would like to take the processor bus
/DACK	The DMA controller that acknowledges the disk interface IC's DMA request
/DMACS	Selects the DMA controller during PIO transfers
/DREQ	The disk interface IC that makes a DMA request to the DMA controller
/DS	Signals that data may be moved into or out of the DMA controller on the multiplexed lines AD0–AD15
/DTACK	Signals that the data-transfer cycle is completed

(continued)

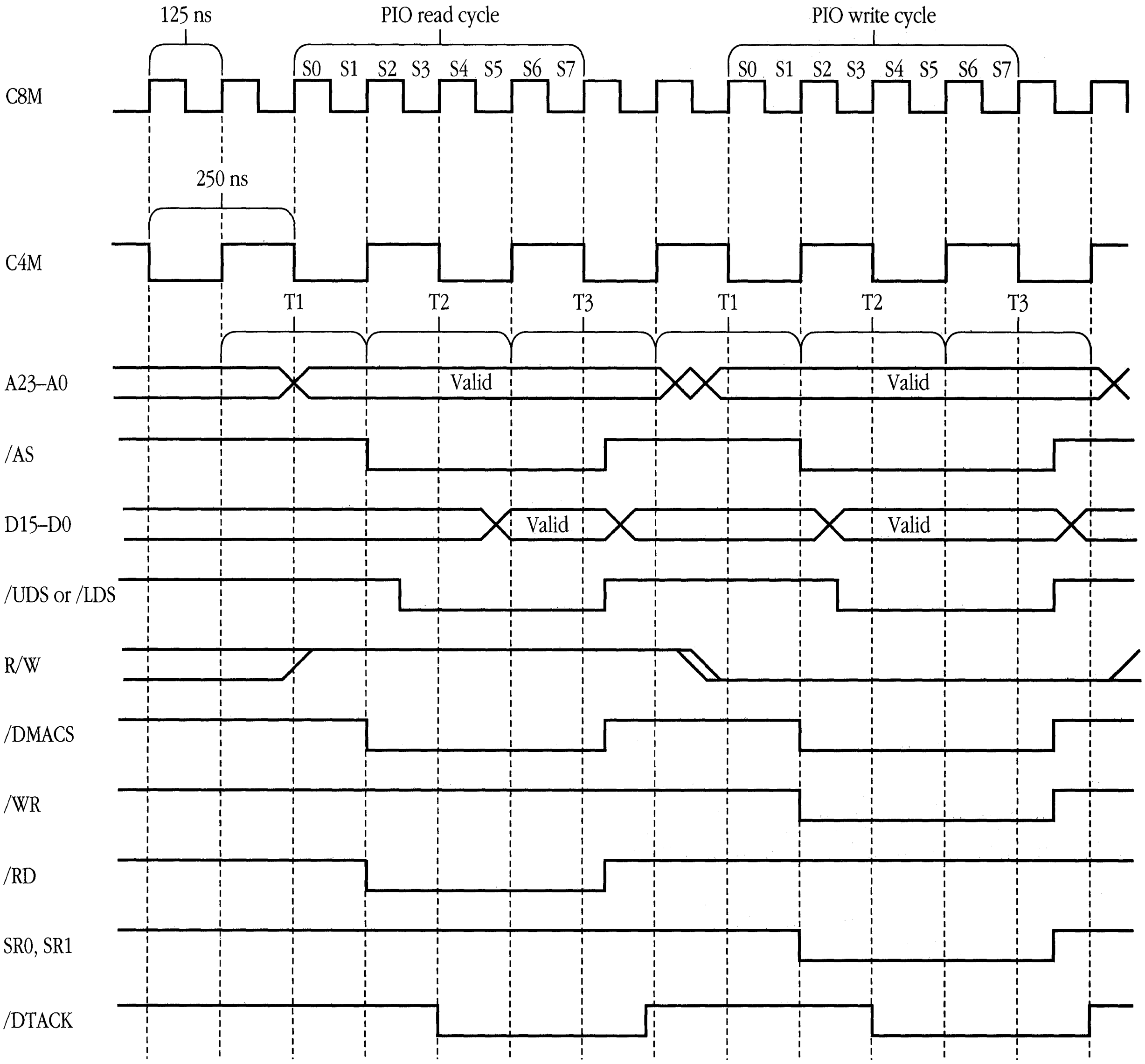
■ **Table 18-1** Bus control signals (continued)

Signal name	Signal description
/EOP	Signals that a disk read or write command has been terminated because the data requested has been transferred
FC0–FC2	The MC68000 processor status codes; serve to signal an interrupt acknowledge cycle when they are all asserted high
/LDS	Indicates that valid data is on the data bus D0–D7
R/W	Defines a cycle to be a read or a write cycle
/UDS	Indicates that valid data is on the data bus D8–D15

Programmed I/O operations

All control information is passed to the disk controller and all status information is transferred to the MC68000 using programmed I/O (PIO) transfers (DMA is used for *data* transfer). The MC68000 host initiates the transfer by asserting signals /AS, R/W, /UDS, and /LDS. Data is then transferred and /DTACK is asserted by the BBU gate array of the Macintosh SE. The PALs decode the address from address lines A18–A23 and thus select either the disk interface IC or the DMA controller to read or write data. Control of the R/W signal determines whether the cycle is a read or a write cycle. See Figure 18-2 for signal timing.

■ **Figure 18-2** Disk controller PIO timing



DMA operations

All data information is transferred to or from the host (MC68000 or a coprocessor) using DMA transfers. After all control information is written to set up both the disk interface IC and the DMA controller, the DMA operation begins.

The disk controller IC requests each DMA transfer, via the signal /DREQ, and that request is funneled through the DMA controller and through the PAL control logic. A bus request is then made, and after the current bus operation has been completed the MC68000 asserts the signal /BG (bus grant). The PAL logic recognizes /BG and waits for any current bus operation to be completed before it signals the disk controller IC to begin a DMA.

As soon as it has taken over the bus, the DMA controller gates the target DMA address onto the lines AD15–AD0 and the lines A23–A16. See Figure 18-1. Using the signal ALE as a reference, the PAL interface logic latches the address from AD15–AD0 with the signal /ADDR. Because all DMA data-transfer operations must be synchronized to the signal /PMCYC (processor memory cycle), the PALs wait for /PMCYC to go low, inserting wait states in the transfer cycle of the DMA controller.

◆ *Note:* The /PMCYC signal is used to synchronize all processor-bus activity. The disk controller waits for /PMCYC to go low before beginning a bus cycle. The signals /AS, /UDS, /LDS, and R/W are not asserted until /PMCYC goes low. The memory timing of the Macintosh SE is synchronized to /PMCYC. Detailed timing information for the /PMCYC signal is provided in Chapter 13. The disk controller is designed to present timing as similar to the MC68000 as possible during a bus cycle (/PMCYC low). /PMCYC goes high during state 0 (S0) of the MC68000 timing and during video memory accesses.

The PALs then assert /AS, /UDS, /LDS, and R/W after gating the address onto the address bus. If a processor read operation (processor reading from the disk interface card) is requested, the PALs gate the data to the correct byte of the data bus from the disk interface IC and generate the proper disk controller read signal. If a processor write operation (processor writing to the disk interface card) is requested, the PALs turn on the correct transceiver to write the data and assert the proper write signal to the disk controller IC.

Address allocation

The disk controller card's device select space ranges from $80 0000 through $8F FFFF and is divided into four blocks. From $80 0000 through $83 FFFF the main status register within the disk controller can be read. A write to this address turns on the signal RST (resets the disk controller). From $84 0000 through $87 FFFF, control, status, and data information may be read from or written to the disk controller data register. Writing in the area $88 0000 through $8B FFFF turns on the drive motor; reading in this area turns both the motor and RST off. The DMA controller is read to or written from via the addressing range $8C 0000 through $8F FFFF. See Table 18-2.

■ **Table 18-2** Device select decode addresses

Decode address range	Device selected and action resulting
$80 0000–$83 FFFF	Reads from the main status register of the disk controller IC and reads an additional status register. The main status register is on the least significant byte, and the additional status register is on the most significant byte. A write to the main status register resets the disk controller IC.
$84 0000–$87 FFFF	Reads or writes control, status, and data information to the data register in the disk controller IC.
$88 0000–$8B FFFF	A write turns drive motor to on; a read turns motor and controller's reset signal off. (Interrupts are enabled when the motor is on.)
$8C 0000–$8F FFFF	Reads from or writes to DMA controller.

Data is normally read from and written to the disk controller card with MC68000 `MOVE.B` instructions. Additional status information may be obtained by reading anywhere in the addressing range $80 0000 through $83 FFFF using `MOVE.W` instructions.

The status register within the disk controller IC may be read with a `MOVE.B` instruction in the address range $80 0000 through $83 FFFF.

The data register within the disk controller IC may be read or written with a `MOVE.B` instruction in the address range $84 0000 through $87 FFFF. It is through the data register that commands, data, and the contents of status registers 0 through 3 are passed. Any disk operation is initiated by passing the several commands required to the disk controller IC via this register.

The read track operation allowed by the disk controller IC is supported on this disk controller. After the execute portion of any operation is completed, the disk controller IC may give back status information in status registers 0 through 3.

Additional status information may be read with a MOVE.W instruction in the address space $80 0000 through $83 FFFF.

The DMA controller is given commands via the chain control table that exists in Macintosh SE RAM. The address of this table is loaded into the chain address register before a chain load command is given to the DMA controller. The chain control table consists of values needed by the DMA controller to transfer data.

Upon receiving a chain load command, the DMA controller loads its registers from the chain control table. After the registers are loaded, the DMA controller is ready to transfer data. Data transfers are then initiated, byte by byte, by the disk controller IC.

Part III Application-Specific Expansion Interfaces

About Part III

Application-specific expansion interfaces are the subject of Part III of this book. These are expansion interfaces that do not fall into the NuBus or processor-direct slot expansion categories, but are designed with a singular, specific purpose in mind. The information in Part III will help you design unique expansion cards that satisfy the requirements of these application-specific interfaces. Part III contains five chapters, and is organized in the same fashion as Part I and Part II.

Chapter 19 gives a hardware overview of the Macintosh computers that provide an application-specific expansion interface but that don't provide a NuBus or a processor-direct expansion interface. Included in this category are the Macintosh Classic, Macintosh Classic II, Macintosh PowerBook 100, Macintosh PowerBook 140, and Macintosh PowerBook 170 computers. Features of each machine are listed, block diagrams are shown, and RAM access, ROM access, and device I/O are discussed briefly.

Chapter 20 discusses the RAM expansion cards for the Macintosh Portable, the Macintosh Classic, the PowerBook 100, the PowerBook 140, and the PowerBook 170 computers. It includes information about address space, RAM expansion connector pinouts and signals, and physical design guides for each computer. It also includes RAM access timing diagrams for the portable Macintosh computers.

Chapter 21 provides information about the ROM expansion interfaces for the Macintosh Portable and Macintosh Classic II computers. It includes information about the ROM expansion address space, the electrical description, the physical design guide, design considerations, and EDisks for the Macintosh Portable. For the Macintosh Classic II, topics include the electrical description of the ROM expansion connector, the ROM expansion address space, the physical design guide, and the power budget for the ROM expansion card.

Chapter 22 focuses on the modem expansion capabilities of the Macintosh Portable, the PowerBook 100, the Powerbook 140, and the PowerBook 170 computers. Topics include the modem card hardware interface, the electrical characteristics of the modem connector, the physical design guide, the modem power-control interface, and the modem operation. Also included in this chapter is reference information for communications standards for modem operation.

Chapter 23 describes the cache memory expansion capability of the Macintosh IIci computer. It includes a description of how the cache works, information on using the cache and accessing memory, cache connector pinouts, signal descriptions, load/drive capabilities, and electrical and mechanical guidelines for designing a cache memory card.

Chapter 19 Application-Specific Expansion Interfaces for Macintosh Computers

This chapter provides an overview of the structure and organization of a third type of Macintosh computer—one that does not provide a NuBus or a general processor-direct expansion interface. These computers provide application-specific expansion interfaces, such as RAM, ROM, modem, and cache expansions. Included in this category are the Macintosh Classic, the Macintosh Classic II, the Macintosh PowerBook 100, the Macintosh PowerBook 140, and the Macintosh PowerBook 170 computers.

The Macintosh Portable and the Macintosh IIci also provide application-specific expansion interfaces. However, because the Macintosh IIci also offers the NuBus expansion, it is described in Chapter 1, "Overview of Macintosh Computers With the NuBus Interface." Likewise, the hardware overview for the Macintosh Portable is presented in Chapter 12, "Overview of Macintosh PDS Computers," because it provides PDS expansion.

Major features

Table 19-1 compares the major features of all Macintosh computers that provide application-specific expansion interfaces.

■ **Table 19-1** Major features of Macintosh computers with application-specific expansions

Feature	Macintosh Classic	Macintosh Classic II	PowerBook 100	PowerBook 140	PowerBook 170
Processor	MC68000 24-bit address bus 16-bit data bus	MC68030 24-bit address bus 16-bit data bus	MC68HC000 24-bit address bus 16-bit data bus	MC68030 32-bit address bus 32-bit data bus	MC68030 32-bit address bus 32-bit data bus
Processor clock	7.8336 MHz	15.6672 MHz	15.6672 MHz	15.6672 MHz	25 MHz
Coprocessor	Not applicable	Floating-point unit (FPU) available on expansion slot	Not applicable	Not applicable	68882 FPU
Memory management	Not applicable	The MC68030 includes a built-in MMU	Not applicable	The MC68030 includes a built-in MMU	The MC68030 includes a built-in MMU
RAM	1 MB, expandable to 4 MB	2 MB, expandable to 10 MB	1 or 2 MB of PSRAM (pseudo-static RAM); expansions of 2, 4 or 6 MB	2 MB of PSRAM; expansions of 2, 4, or 6 MB	2 MB of PSRAM; expansions of 2, 4, or 6 MB
ROM	512 KB, expandable to 2 MB	512 KB, expandable to 4 MB	256 KB	1 MB	1 MB
Expansion capabilities	RAM	FPU/ROM	RAM, modem	RAM, modem	RAM, modem
Input device interface	One Apple Desktop Bus (ADB) port	One ADB port; microphone jack for sound input	One ADB port	One ADB port; microphone jack for sound input	One ADB port; microphone jack for sound input

(continued)

■ **Table 19-1** Major features of Macintosh computers with application-specific expansions (continued)

Feature	Macintosh Classic	Macintosh Classic II	PowerBook 100	PowerBook 140	PowerBook 170
Serial ports	Two mini 8-pin connectors	Two mini 8-pin connectors supporting RS-422	One external mini 8-pin connector supporting RS-422; internal 20-pin modem connector	Two mini 8-pin connectors; internal 20-pin modem connector	Two mini 8-pin connectors; internal 20-pin modem connector
Floppy disk support	Super Woz Integrated Machine (SWIM) controls one internal 1.4 MB SuperDrive; one optional external port	SWIM controls one internal 1.4 MB, 3.5" SuperDrive and one optional external floppy disk drive	SWIM controls one optional external floppy disk drive, via the HDI-20 (High-Density Interface, 20-pin) connector	SWIM controls one internal 1.4 MB 19 mm SuperDrive	SWIM controls one internal 1.4 MB 19 mm SuperDrive
Hard disk	Optional internal 40 MB SCSI hard disk	Internal 40 MB SCSI hard disk; optional external SCSI hard disk	Internal 20 MB SCSI hard disk; optional external SCSI hard disk	One internal 20 MB, 2.5" SCSI hard disk	One internal 40 MB, 2.5" SCSI hard disk
SCSI port	One internal 50-pin; one external DB-25	One internal 50-pin; one external DB-25	One internal 40-pin; one external HDI-30 connector	One internal HDI-30 connector; one external HDI-30 connector	One internal HDI-30 connector; one external HDI-30 connector
Sound	Monaural sound via standard Macintosh sound chip	Monaural sound output; Digitally Filtered Audio Chip (DFAC) provides digital recording feature	Apple Sound Chip (ASC)	Enhanced ASC and DFAC	Enhanced ASC and DFAC
Video display	Built-in 9" monochrome monitor, 512 × 342 pixels	Built-in 9" monochrome monitor, 512 × 342 pixels	Built-in backlit, flat-panel LCD, 640 × 400 pixels	Built-in backlit, flat-panel Film SuperTwist Nematic LCD, 640 × 400 pixels	Built-in backlit, flat-panel active matrix LCD, 640 × 400 pixels
Battery	Long-life lithium battery backup	Long-life lithium battery backup	2.5-ampere-hour SLA (sealed lead-acid), rechargeable battery; 3 V lithium battery backup	2.8-ampere-hour NiCad rechargeable battery; 3 V lithium battery backup	2.8-ampere-hour NiCad rechargeable battery; 3 V lithium battery backup

Hardware architecture

The following discussion is brief and intended primarily to show the place of each application-specific expansion interface in the machine architecture. For a complete description of hardware operation, see the *Guide to the Macintosh Family Hardware.* Also useful would be the *Macintosh IIsi, LC, and Classic Developer Notes* and the *Macintosh Classic II, Macintosh PowerBook Family, and Macintosh Quadra Family Developer Notes.* Or, if you are interested in a higher-level overview, see the *Technical Introduction to the Macintosh Family.*

The Macintosh Classic is similar in appearance to the Macintosh SE. In fact, the Macintosh Classic computer's main logic board is based on the Macintosh SE design. The Macintosh Classic uses an MC68000 microprocessor running at 7.8336 MHz and provides Macintosh SE performance. The main architectural difference between the two computers is that the Macintosh Classic offers a RAM expansion interface instead of the processor-direct expansion interface of the Macintosh SE.

The Macintosh Classic II computer is also similar in appearance to the Macintosh SE; however, its electrical design is based as much as possible on the Macintosh LC architecture. Instead of the MC68020 processor used by the Macintosh LC, the Macintosh Classic II uses an MC68030 processor that runs at 15.6672 MHz. The Macintosh Classic II offers optional FPU and ROM expansion. For more information, refer to Chapter 21, "ROM Expansion Interface."

The PowerBook family of computers are the latest Macintosh portable computers. The Macintosh PowerBook 100 is a new notebook-class, battery-operated, portable computer. Its architecture is based on that of the Macintosh Portable. Both use the MC68HC000 microprocessor and run at 15.6672 MHz. However, the PowerBook 100 does not include an internal floppy disk drive or a processor-direct expansion slot like the Macintosh Portable. The PowerBook 100, like the Macintosh Portable, however, does provide RAM, ROM, and modem expansion interfaces.

The Macintosh PowerBook 140 and Macintosh PowerBook 170 computers are also based on the original Macintosh Portable; however, they are considerably smaller. The PowerBook 140 is a slightly less powerful version than the PowerBook 170. Both use the MC68030 processor, but the PowerBook 170 runs at 25 MHz and the PowerBook 140 runs at 15.6672 MHz. The PowerBook 170 also provides an FPU, whereas the PowerBook 140 does not. Both the PowerBook 140 and the PowerBook 170 computers provide RAM and modem expansion interfaces.

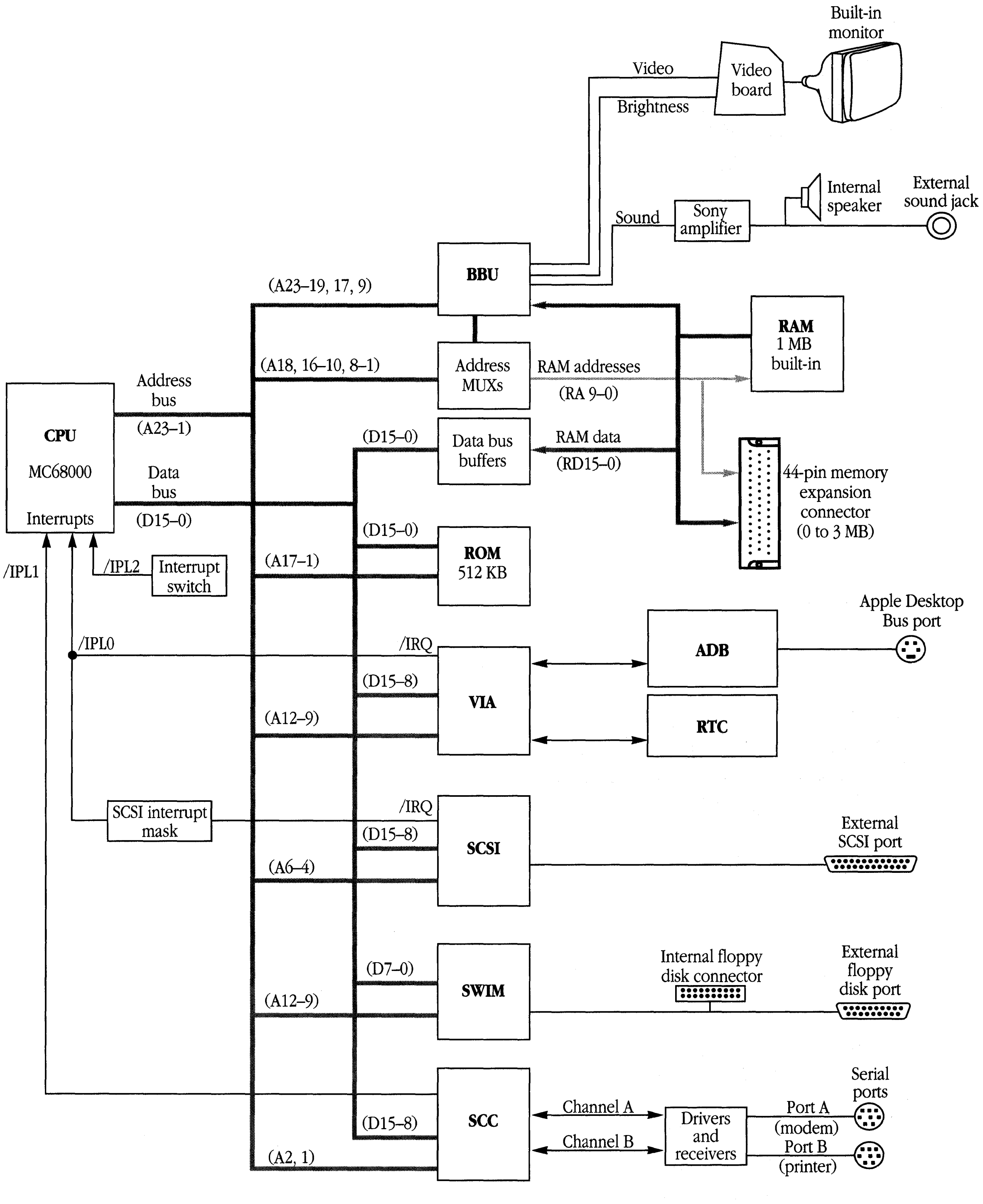
Block diagrams of the Macintosh Classic, the Macintosh Classic II, the PowerBook 100, the PowerBook 140, and the PowerBook 170 are shown in Figures 19-1 through 19-4.

The computers that provide application-specific expansion interfaces contain several common circuits, including RAM, ROM, and some I/O chips that enable the microprocessor to communicate with external devices. The following is a brief description of these I/O chips:

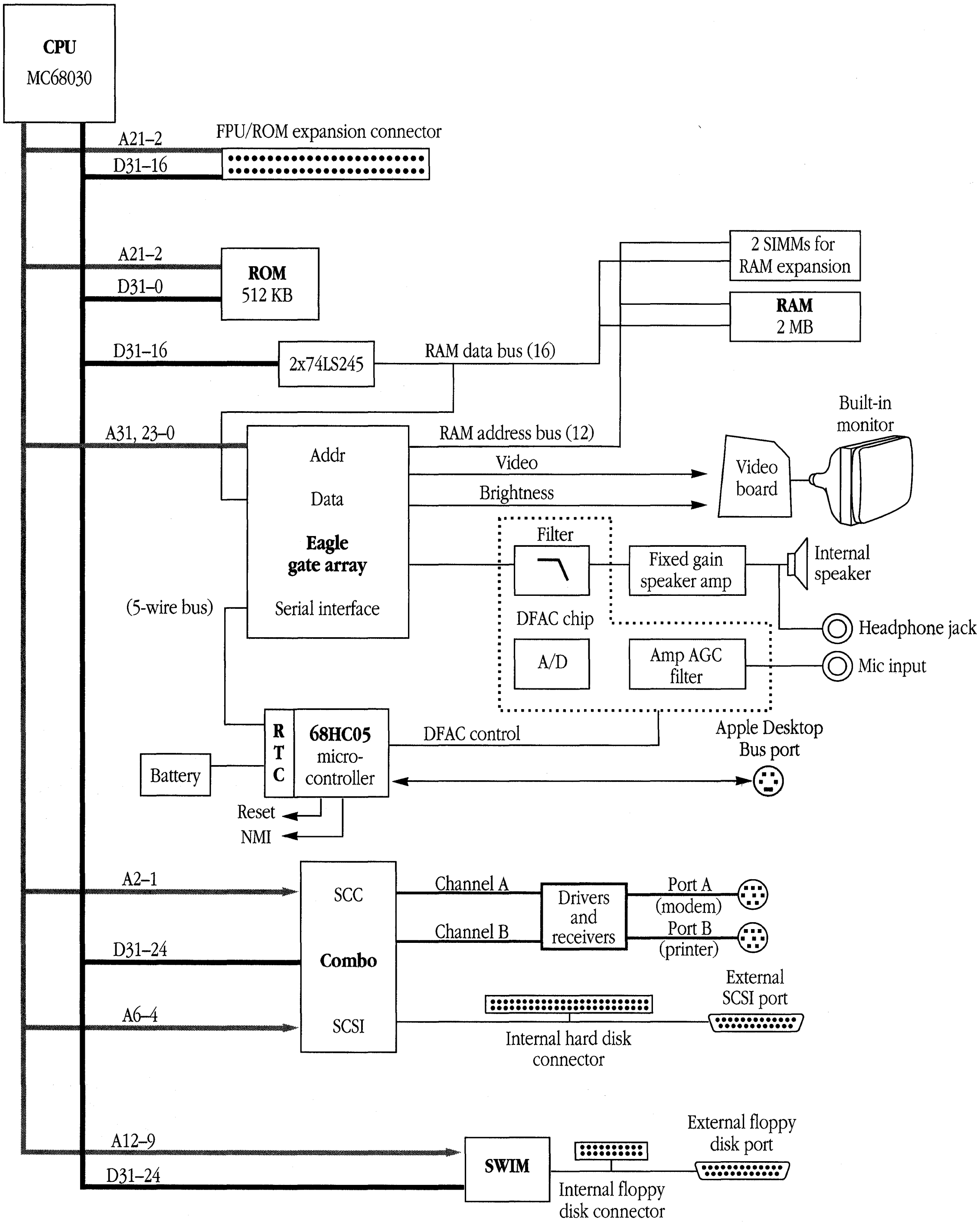
- Every Macintosh computer except the Macintosh Classic II has one or two Versatile Interface Adapter (VIA) chips. The Macintosh Classic, the PowerBook 100, the PowerBook 140, and the PowerBook 170 each have one VIA chip. The VIA in the Macintosh Classic, the PowerBook 140, and the PowerBook 170 supports the real-time clock (RTC). The VIA in the PowerBook 100 provides the communication interface between the processor and the Power Manager IC as well as interrupts for a number of internal functions. VIA functions in the Macintosh Classic II are provided in the Eagle gate array, described later in this section.
- A SCSI (Small Computer System Interface) chip provides high-speed parallel communication with internal or external devices such as hard disks. A Serial Communications Controller (SCC) provides for high-speed, asynchronous serial communication (also synchronous modem support). These two chips are used only in the Macintosh Classic. The Combo chip, used in the Macintosh Classic II, the PowerBook 100, the PowerBook 140, and the PowerBook 170, combines the functions of the SCC and SCSI chips. This device is completely software compatible with the SCC and SCSI chips it replaces. It controls the two ports for serial communications and SCSI ports for connection to hard disks.
- An Apple custom chip controls both internal and external floppy disk drives. The Macintosh Classic, the Macintosh Classic II, and the PowerBook-family computers all use the SWIM (Super Woz Integrated Machine) chip to control their floppy drives.
- The Macintosh Classic includes an Apple custom chip called the BBU (Bob Bailey Unit), also used in the Macintosh SE, for video and sound control and for generating device-select signals. The computers in the PowerBook family use the Apple Sound Chip (ASC) to control stereo sound. The Digitally Filtered Audio Chip (DFAC) is a custom IC that does the analog processing functions for the sound system. It is used in the Macintosh Classic II, the PowerBook 140, and the PowerBook 170.
- The Power Manager IC was introduced in the Macintosh Portable to control the distribution of power to all I/O devices. It is also used in the PowerBook-family computers. In the PowerBook 140 and PowerBook 170, however, functions such as the real-time clock (RTC) and the PRAM have been removed from the Power Manager microprocessor and are now provided by an RTC chip.
- The PowerBook-family computers include two general logic unit chips, the CPU GLU and the Miscellaneous GLU. These chips generate CPU clocks and data acknowledgment signals, provide PSRAM chip select and refresh, do modem–serial port multiplexing, and provide the interfaces to the rest of the system.

- A custom IC, the Eagle gate array, is included in the Macintosh Classic II computer. In addition to providing the general logic unit functions, the Eagle provides timing, video generation, memory mapping, sound, and clock generation. The Eagle functionally replaces the RBV (RAM-based video) chip, the MDU (Memory Decode Unit), the VIA chips, and the ASC.
- The DDC (Display Driver Chip) provides the interface to the LCD in the PowerBook-family computers. Its function is similar to that of the video chip used in the Macintosh Portable, except that the DDC supports FSTN (Film SuperTwist Nematic) displays as well as active matrix (AM) displays. The DDC generates horizontal and vertical synchronization pulses, and all other signals necessary to make the flat-panel display work.

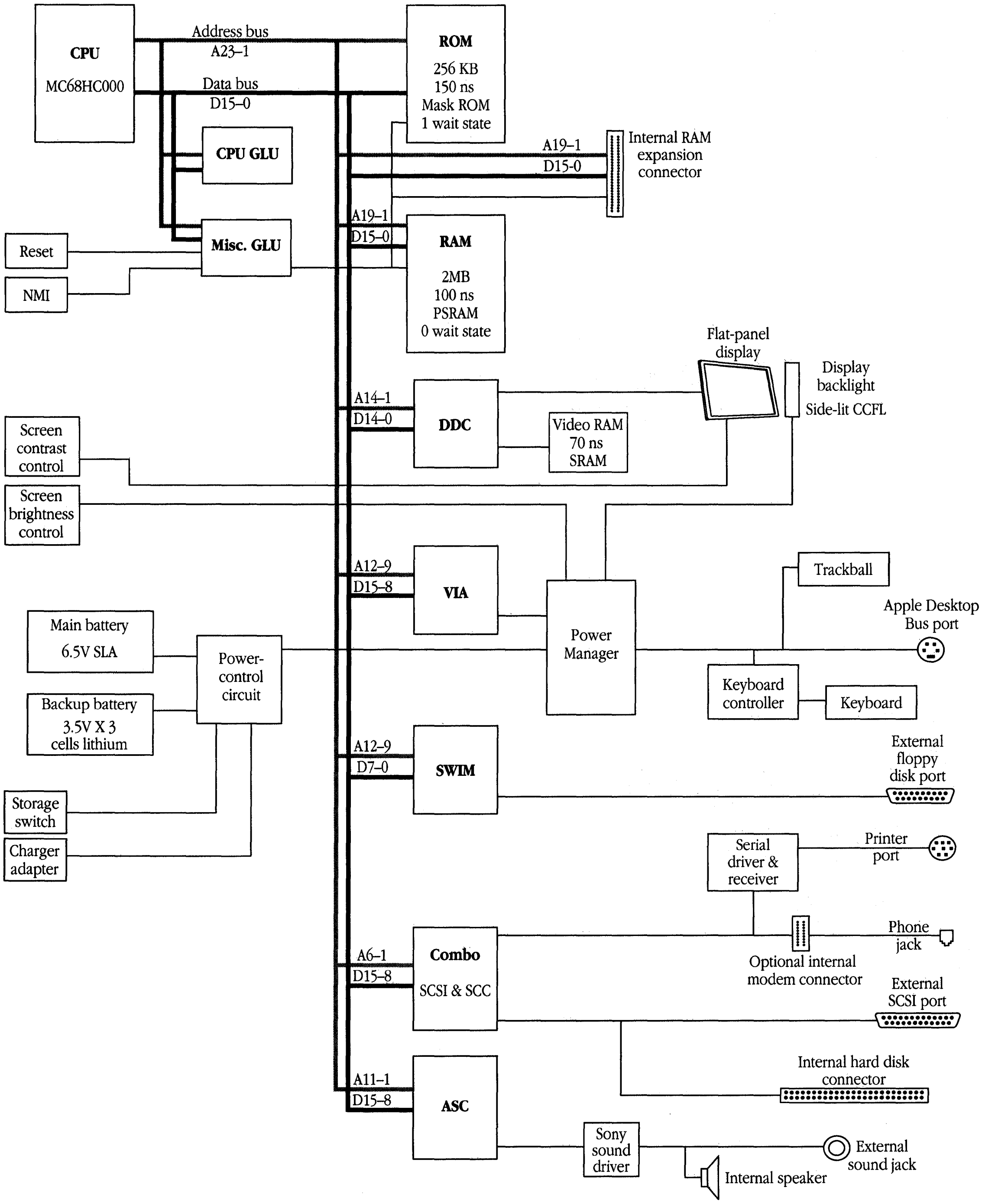
■ **Figure 19-1** Block diagram of the Macintosh Classic computer



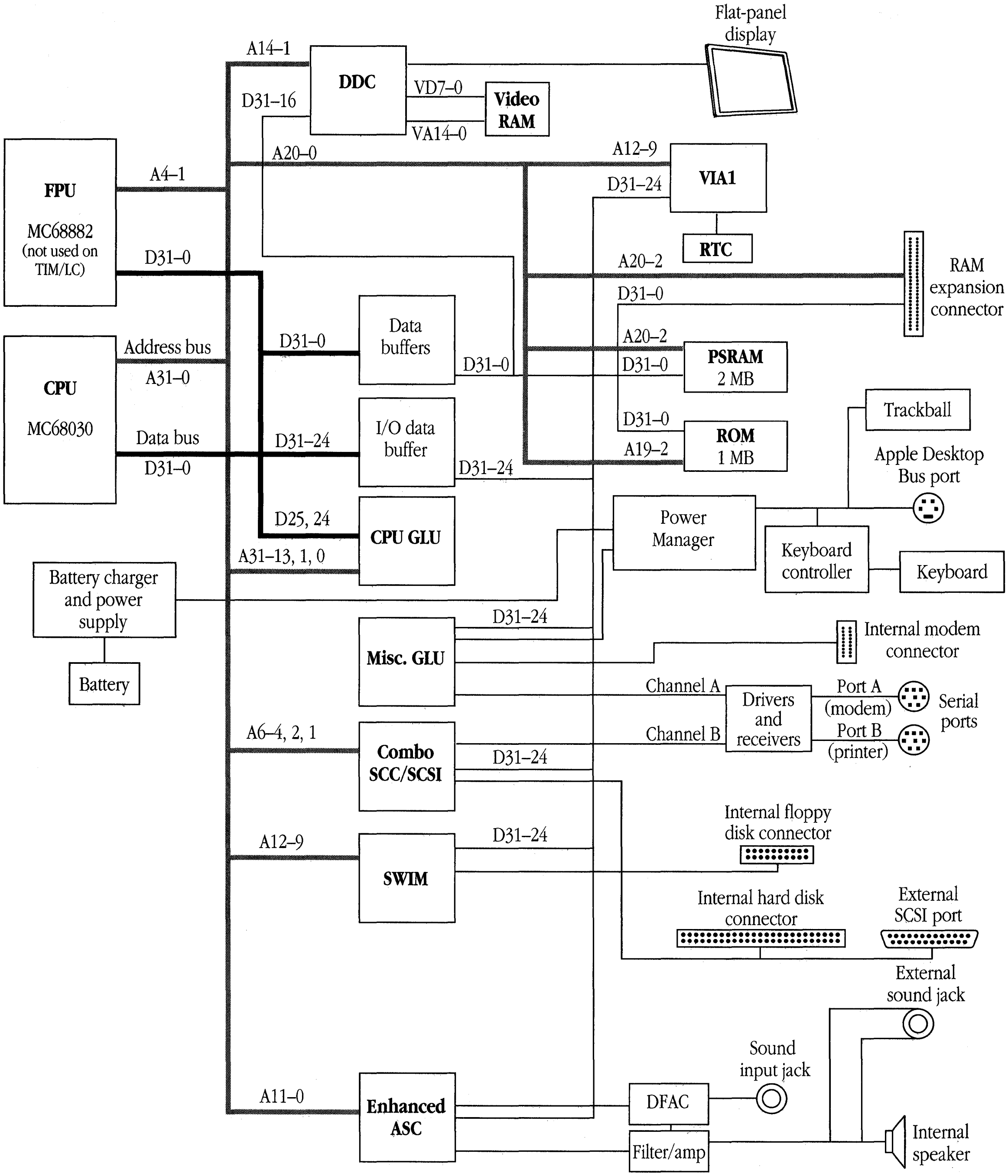
■ **Figure 19-2** Block diagram of the Macintosh Classic II computer



■ **Figure 19-3** Block diagram of the Macintosh PowerBook 100 computer



■ **Figure 19-4** Block diagram of the Macintosh PowerBook 140 and Macintosh PowerBook 170 computers



RAM

RAM is the working memory of the system. In the Macintosh Classic computer, address space from $00 0000 through $3F FFFF is reserved for RAM. In the PowerBook 100, address space from $00 0000 through $7F FFFF is reserved for RAM. Address space $0000 0000 through $7FFF FFFF is reserved for RAM in the PowerBook 140 and the PowerBook 170 computers. In the Macintosh Classic II, address space from $00 0000 through $9F FFFF is reserved for RAM. The actual amount of address space used depends upon the amount of RAM available in the system.

The first 1024 bytes of RAM (addresses $00 0000 through $00 03FF) are used as storage for exception vectors; these are the addresses of the routines that gain control whenever an exception such as an interrupt or a trap occurs. The first 256 bytes are reserved for use by the operating system, and the remainder are allocated for use by applications. RAM contains the system heap, a copy of parameter RAM, various global variables and trap handlers, application heaps, the stack, and other information used by applications.

ROM

ROM is the system's permanent read-only memory. When the Macintosh is first turned on, a second image of ROM appears at $00 0000, so that ROM can supply the processor with the exception vectors. Following the first access to the normal address ranges of ROM or the SCSI controller, the image of ROM at $00 0000 is replaced by RAM.

The base address is available in the global variable `ROMBase`. ROM contains the routines for the User Interface Toolbox and the Macintosh Operating System, and the various system traps.

The PowerBook 140 and PowerBook 170 computers come equipped with 1 MB of ROM on the main logic board. The ROM will support new features in the PowerBook 140 and the PowerBook 170, such as power cycling, the Power Manager, modem support, the backlit display, and the memory controller.

Device I/O

All Macintosh computers use memory-mapped I/O, which means that each device in the system is accessed by reading from or writing to specific locations in the address space of the computer. The address space reserved for the device I/O contains blocks devoted to each of the devices within the computer. Each device contains logic that recognizes when it's being accessed and allows the device to respond in the appropriate manner. For more information about the address space for each computer covered in this section, please refer to the next three chapters.

Chapter 20 **RAM Expansion Interface**

This chapter describes the RAM expansion interfaces provided in the Macintosh Portable, the Macintosh Classic, the PowerBook 100, the PowerBook 140, and the PowerBook 170 computers. It describes the electrical characteristics of each RAM expansion connector and physical specifications for installing a RAM expansion card in each type of Macintosh computer.

Macintosh Portable RAM expansion

The RAM expansion interface in the Macintosh Portable computer is designed to support up to 5 MB of CMOS static RAM. The Macintosh Portable comes equipped with a main memory consisting of 1 MB of permanent RAM soldered to the main logic board. Because of the increasing size of application programs, the Macintosh Portable is designed to accommodate an expansion card that will provide up to 4 MB of additional RAM for the system.

There are actually two versions of the Macintosh Portable—the original Macintosh Portable and the backlit Macintosh Portable. The backlit version is slightly different from the original Macintosh Portable. Along with providing a backlit active matrix display, some other improvements were made to the backlit Macintosh Portable:

- PSRAM (pseudostatic RAM) replaces the SRAM (static RAM) used in the original Macintosh Portable.
- A shredded, more flexible cable provides easier access to the processor-direct expansion slot.
- The RAM expansion connector has been keyed, and the pinout has been slightly changed (two pins are different) from that of the original Macintosh Portable.
- Backlighting, which can be controlled by either a user (via the control panel's Portable CDev) or third-party application software, has been added to improve the backlit Macintosh Portable computer's display quality.

Like the Macintosh Portable, the backlit version is designed to accommodate a RAM expansion card that can provide up to 4 MB of additional RAM for the system.

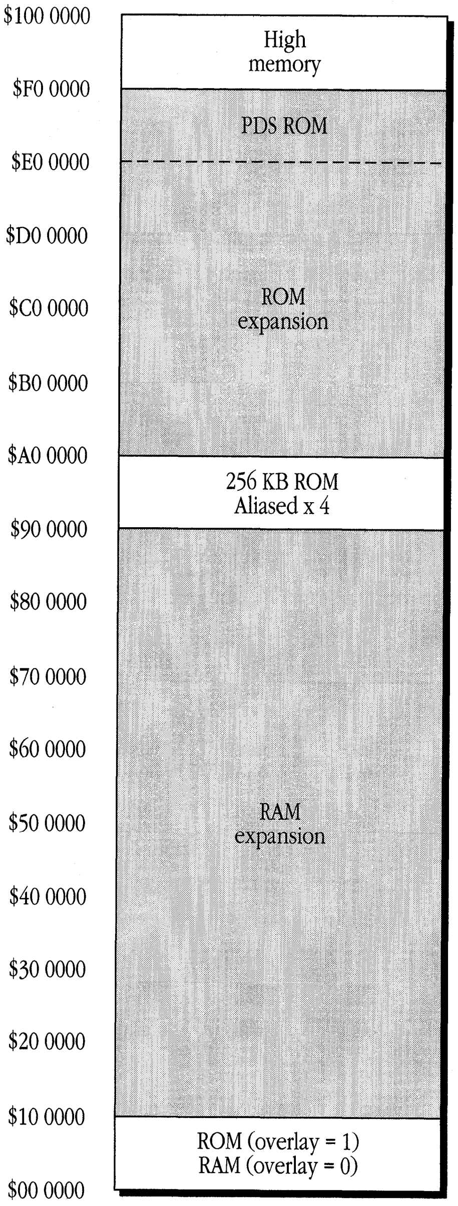
Differences between the original Macintosh Portable and the backlit Macintosh Portable are pointed out in this section. Where no mention is made of differences for a particular feature, you can assume the feature is identical in both machines.

Macintosh Portable RAM expansion address space

The 1 MB permanent RAM memory is arranged as a 512 K × 16-bit array. This RAM array is located between addresses $00 0000 and $0F FFFF in the Macintosh Portable memory map (Figure 20-1), and is overlaid by the system ROM after a system reset and before the first ROM access.

The 8 MB space between addresses $10 0000 and $8F FFFF is reserved for RAM expansion. It is possible to expand RAM up to 8 MB; however, the zero wait state /DTACK signal is generated only for the first 5 MB of RAM address space. (The 5 MB includes 1 MB on the main logic board and 4 MB on a RAM expansion card.) You can design your expansion card for any of a number of possible configurations of additional RAM. For example, Apple has designed a 1 MB RAM expansion card. The 1 MB expansion card is arranged as a 512 K × 16-bit array and is located between addresses $10 0000 and $1F FFFF in the memory map. You could design a 3 MB expansion card with memory arranged as a 1.5 M × 16-bit array. This configuration would be located between addresses $10 0000 and $3F FFFF in the memory map. The access time and cycle time for each of these configurations are 100 ns. The size of the RAM array is determined by the type of RAM chips you use. When your card is installed in the Macintosh Portable, the memory array is always available and, unlike permanent main memory, is unaffected by the state of the overlay bit.

■ **Figure 20-1** Macintosh Portable memory map

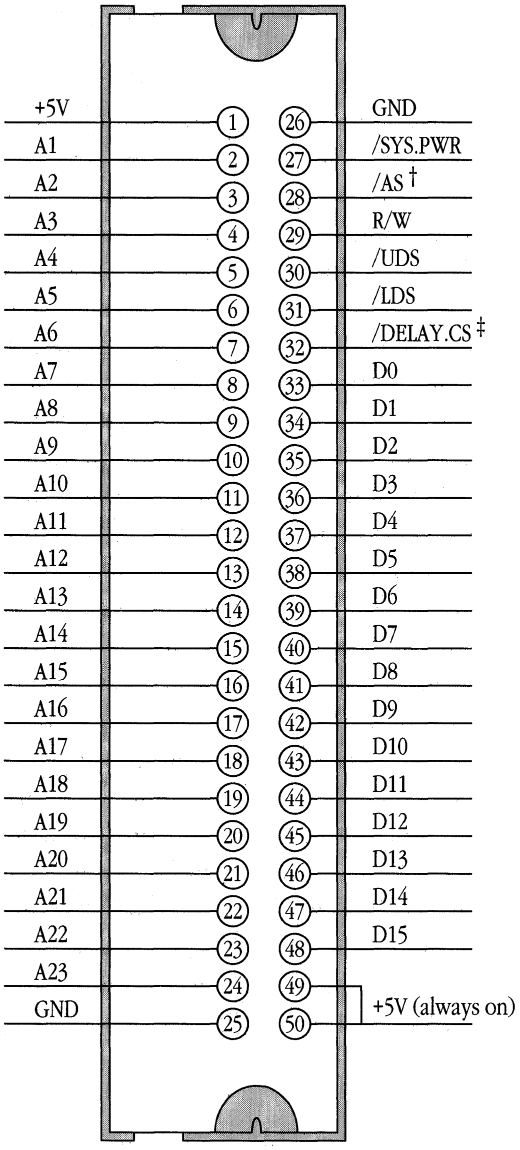


RAM expansion cards for the Macintosh Portable

Your RAM expansion card connects to the Macintosh Portable through a single 50-pin connector (slot) on the Macintosh Portable main logic board. Chapter 17 provides information about the location of the connectors on the main logic board. Figure 20-2 shows the pinout of the RAM expansion connector.

All necessary address bus, data bus, and control signals from the Macintosh Portable are provided to the expansion card through the RAM expansion connector. Table 20-1 provides the names and descriptions of each signal. Apple uses a custom IC to decode, control, and buffer the signals going to the expansion card. You must remember to include similar circuitry in your expansion card design. Buffering of the address bus and data bus is important to reduce capacitive loading. You should also use CMOS devices, because the maximum current allotted to the RAM expansion connector is only 25 mA.

■ **Figure 20-2** Macintosh Portable RAM expansion connector pinout



† Pin 28 is defined as /RAM.CS in the backlit Macintosh Portable.

‡ Pin 32 is defined as /REFRESH in the backlit Macintosh Portable.

■ **Table 20-1** Macintosh Portable RAM expansion connector signals

Pin number	Signal name	Signal description
1	+5V	+5-volt power supply
2–24	A1–A23	Unbuffered 68HC000 address signals A1–A23
25–26	GND	Logic ground
27	/SYS.PWR	Controls whether the Macintosh Portable is in the operating state or sleep state
28	/AS†	68HC000 address strobe signal
29	R/W	68HC000 read/write signal
30	/UDS	Upper data strobe signal
31	/LDS	Lower data strobe signal
32	/DELAY.CS‡	Signal generated by the CPU GLU chip to put the RAM array into the idle mode
33–48	D0–D15	Unbuffered 68HC000 data signals D0–D15
49–50	+5V	+5-volt power supply

† Pin 28 is defined as /RAM.CS in the backlit Macintosh Portable.
‡ Pin 32 is defined as /REFRESH in the backlit Macintosh Portable.

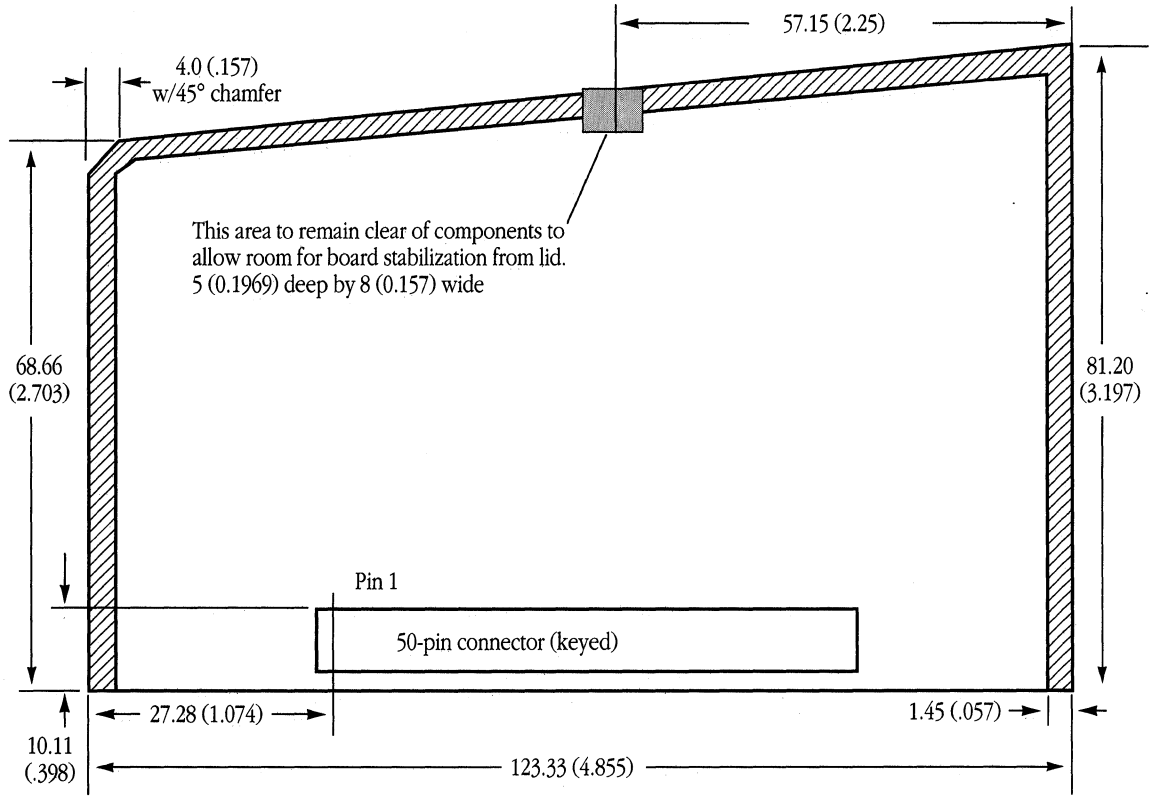
There is one 68HC000 processor wait state when accessing memory locations in the expansion RAM. This access requires a bus cycle of nominally 320 ns. Like permanent RAM, there is no device contention for bandwidth other than the 68HC000 processor; and, because the memory array is static RAM, it does not have to be refreshed, as would be the case for dynamic RAM.

Also, like permanent RAM, the expansion RAM is backed up by the battery when the Macintosh Portable is in the sleep state. This means that the contents of the expansion RAM are retained when the computer is not in use, as long as the battery is charged.

Figure 20-3 is a design guide providing the physical specifications you need to design a RAM expansion card for the Macintosh Portable.

▲ **Warning** Figure 20-3 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change. ▲

■ **Figure 20-3** Macintosh Portable RAM expansion card design guide



Dimensions are in millimeters with inches in parentheses.

RAM expansion cards for the backlit Macintosh Portable

The RAM expansion connector (slot) is physically identical to the connector used on the original Macintosh Portable with the exception that the backlit Macintosh Portable connector is keyed. Electrically, however, there is a slight difference because two of the pins have different assignments from those of the original Macintosh Portable. Table 20-2 lists the pin assignments that have changed.

■ **Table 20-2** RAM expansion connector signal differences for the backlit Macintosh Portable

Pin number	Original Macintosh Portable	Backlit Macintosh Portable
28	/AS	/RAM.CS
32	/DELAY.CS	/REFRESH

The /RAM.CS signal is directly related to the /REFRESH signal. The /RAM.CS signal goes high to signal that reading and writing are no longer valid, just before /REFRESH goes low to refresh the RAM memory. The /REFRESH signal is normally high and goes low (active state) approximately every 16 µs. It remains low for 180 ns, during which time the RAM memory is refreshed.

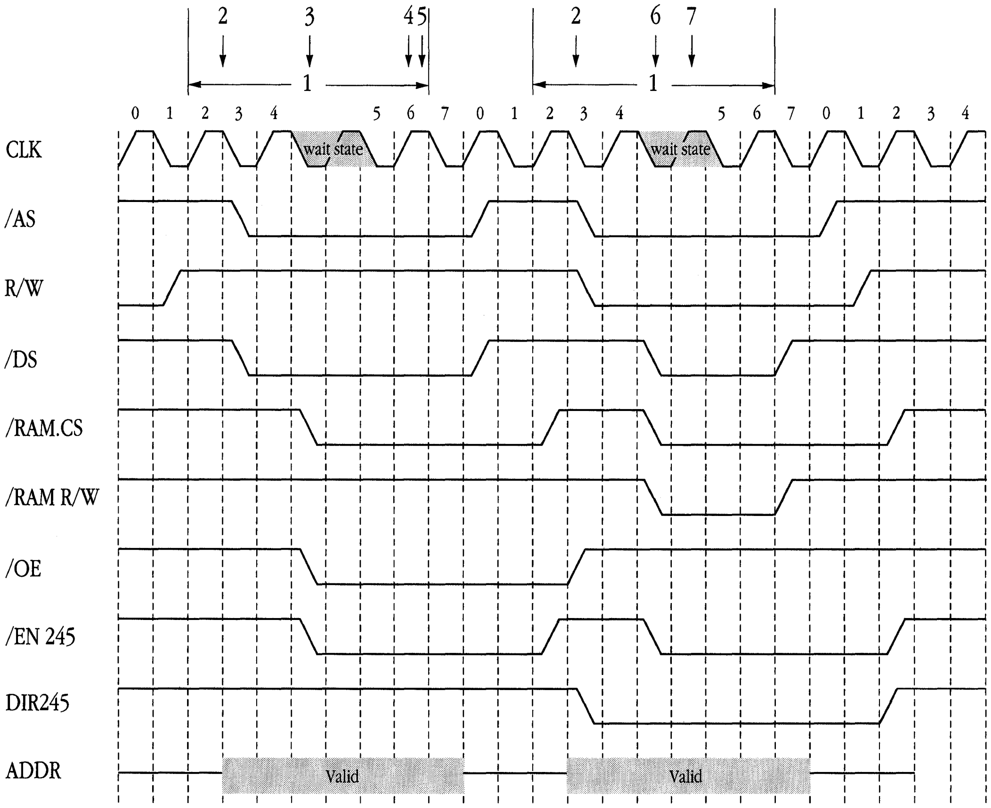
△ **Important** Because an Apple RAM expansion card does not fully decode RAM expansion space, the card must be removed before additional third-party RAM can be added to the backlit Macintosh Portable. It is possible to expand RAM up to 8 MB; however, the zero wait state /DTACK signal and the /REFRESH signal are generated only for the first 5 MB of RAM address space. Apple does not recommend that you use the PDS for RAM expansion; however, if you are planning to design a processor-direct slot (PDS) expansion card for additional RAM, keep in mind that timing, particularly the /DTACK signal, is critical. △

RAM expansion-slot timing for the backlit Macintosh Portable

Unlike the original Macintosh Portable, which required one wait state, the backlit Macintosh Portable requires zero wait states when its processor is accessing memory locations in the expansion RAM. Figure 20-4 shows the RAM expansion-slot timing for the backlit Macintosh Portable.

△ **Important** Inserting an additional 64 wait states between CLK cycles 4 and 5 (see Figure 20-4) when the backlit Macintosh Portable is in idle mode causes the processor timing to effectively slow down to 1 MHz. △

■ **Figure 20-4** RAM expansion-slot timing for the backlit Macintosh Portable



Notes:
1. Total length of cycle read or write 5 x 32 ns = 160 ns
2. Delay clock to /AS = 30 ns
3. Delay /AS to /RAM.CS = 10 ns
4. Delay through 74ac245 buffer = 7 ns
5. Setup before s7 = 5 ns
6. Delay s4 to /DS falling = 30 ns
7. Delay /DS to RAM R/W falling = 20 ns

Read timing:		Write timing:	
Cycle	160	Cycle	160
/AS	30	/AS	64
/RAM.CS	10	/RAM.CS	30
buffer	7	buffer	20
setup	5	setup	100
RAM	100	RAM	60
Margin	8 ns	Margin	30 ns

Important:
When the Portable is in idle mode, an additional 64 wait states are added between clk 4 and 5, effectively slowing the CPU timing down to 1 MHz. During normal operation these wait states are not present.

Design considerations for RAM expansion in the backlit Macintosh Portable

You must design your RAM expansion card to operate at zero wait states if you expect it to work in the backlit Macintosh Portable computer.

Because of different pin functions and the requirement of zero wait states, a RAM card designed for the backlit Macintosh Portable must have a keyed connector to prevent it from being installed in the original Macintosh Portable. However, a RAM expansion card designed for the original Macintosh Portable can be installed in the backlit Macintosh Portable. For the card to function correctly in both machines, it must be designed to run at zero wait states, and must not include a connection to the /DELAY.CS signal (/REFRESH for backlit Macintosh Portable).

△ **Important** Developers of expansion cards for the processor-direct slot must remember that /EXT.DTACK will no longer delay /DTACK for accesses to the first 5 MB of address space ($00 0000 through $4F FFFF). RAM or other devices responding to addresses in the first 5 MB are required to run at full speed with no wait states. △

Macintosh Classic RAM expansion

The Macintosh Classic is equipped with 1 MB of RAM soldered to the main logic board. You can design RAM expansion cards that can provide 2, 2.5, or 4 MB of additional memory.

This section describes the Macintosh Classic address space and the electrical and mechanical information that you need to design a RAM expansion card.

Macintosh Classic RAM expansion address space

The 1 MB of RAM in the Macintosh Classic consists of eight 256 K × 4 dynamic random-access memory (DRAM) chips soldered to the main logic board. This RAM array is located between addresses $00 0000 and $0F FFFF in the Macintosh Classic memory map. It is overlaid by the system ROM after a system reset and before the first ROM access.

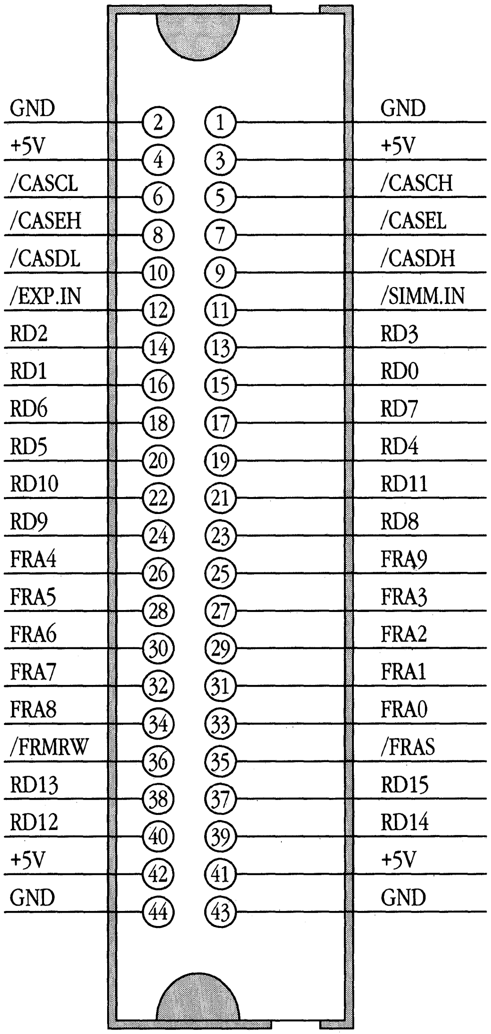
The 8 MB space in the Macintosh Classic between addresses $10 0000 and $3F FFFF is reserved for RAM expansion. Because the Macintosh Classic is based on the design of the Macintosh SE, the description of the Macintosh SE address space provided in Chapter 13 is the same for the Macintosh Classic.

Electrical description of the RAM expansion cards for the Macintosh Classic

Figure 20-5 gives the pinout for the 44-pin RAM expansion connector for the Macintosh Classic. Table 20-3 provides the pinout and signal descriptions of the memory expansion connector.

A Programmed Array Logic (PAL) integrated circuit on the main logic board sends the necessary column address strobe (CAS) signals to the expansion connector. Pins 12 and 11 on the connector indicate to the PAL whether an expansion card is installed and, if one is, whether it contains SIMMs. Pin 12 (/EXP.IN) is the expansion input pin. This pin must be grounded to indicate that a memory expansion card is installed. Pin 11 (/SIMM.IN) is the SIMM input pin. If a memory card is installed, this pin must be grounded to indicate that SIMMs are plugged into the card. The status of the /EXP.IN and /SIMM.IN signals also determines which CAS lines are active. Input signals on the expansion connector do not have to be pulled high.

■ **Figure 20-5** Macintosh Classic RAM expansion connector pinout



■ **Table 20-3** Macintosh Classic RAM expansion connector signals

Pin number	Name	Description
1	GND	Ground
2	GND	Ground
3	+5V	+5-volt power
4	+5V	+5-volt power
5	/CASCH	Bank C high-byte column address strobe
6	/CASCL	Bank C low-byte column address strobe
7	/CASEL	Bank E low-byte column address strobe
8	/CASEH	Bank E high-byte column address strobe
9	/CASDH	Bank D high-byte column address strobe
10	/CASDL	Bank D low-byte column address strobe
11	/SIMM.IN	SIMM input indicator
12	/EXP.IN	Expansion input indicator
13	RD3	Data bit 3
14	RD2	Data bit 2
15	RD0	Data bit 0
16	RD1	Data bit 1
17	RD7	Data bit 7
18	RD6	Data bit 6
19	RD4	Data bit 4
20	RD5	Data bit 5
21	RD11	Data bit 11
22	RD10	Data bit 10
23	RD8	Data bit 8
24	RD9	Data bit 9
25	FRA9	Address bit 9
26	FRA4	Address bit 4
27	FRA3	Address bit 3
28	FRA5	Address bit 5
29	FRA2	Address bit 2
30	FRA6	Address bit 6
31	FRA1	Address bit 1
32	FRA7	Address bit 7
33	FRA0	Address bit 0

(continued)

■ **Table 20-3** Macintosh Classic RAM expansion connector signals (continued)

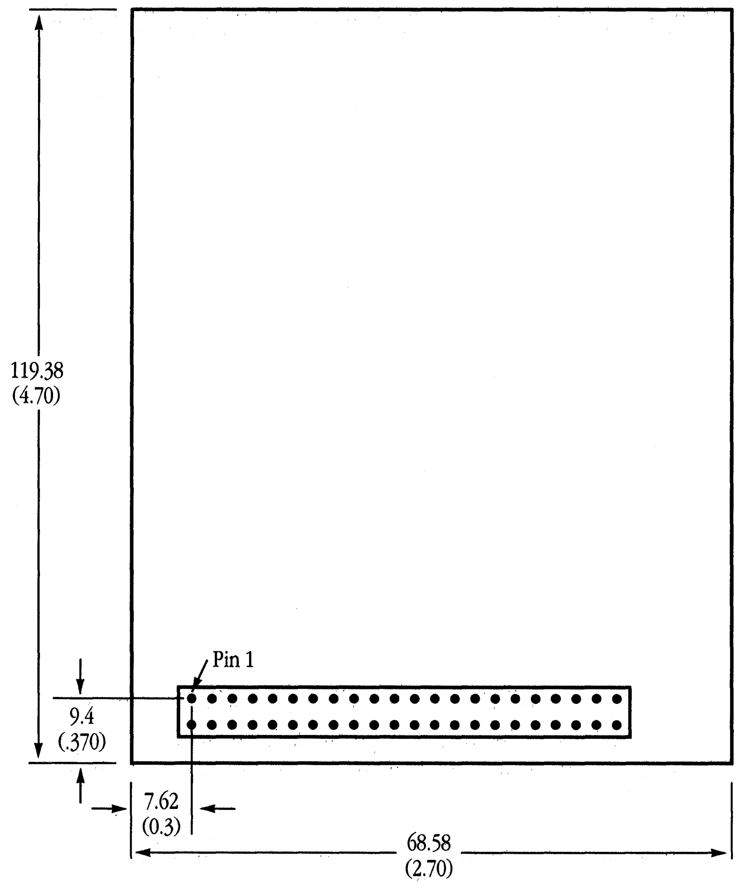
Pin number	Name	Description
34	FRA8	Address bit 8
35	/FRAS	Row address strobe
36	/FRMRW	Read/write
37	RD15	Data bit 15
38	RD13	Data bit 13
39	RD14	Data bit 14
40	RD12	Data bit 12
41	+5V	+5-volt power
42	+5V	+5-volt power
43	GND	Ground
44	GND	Ground

Physical design guide for a Macintosh Classic RAM expansion card

Your RAM expansion card connects to the Macintosh Classic via the 44-pin connector on the main logic board. Figure 20-6 is a design guide showing the maximum allowable card size and the location of pin 1 for installing the expansion card's 44-pin connector. Some available 44-pin connectors are

- Part number 2–532955–4
  AMP, Incorporated
  Harrisburg, PA 17105
- Part number 15–44–6044
  Molex, Incorporated
  Lisle, IL 60532

■ **Figure 20-6** RAM expansion card design guide for the Macintosh Classic

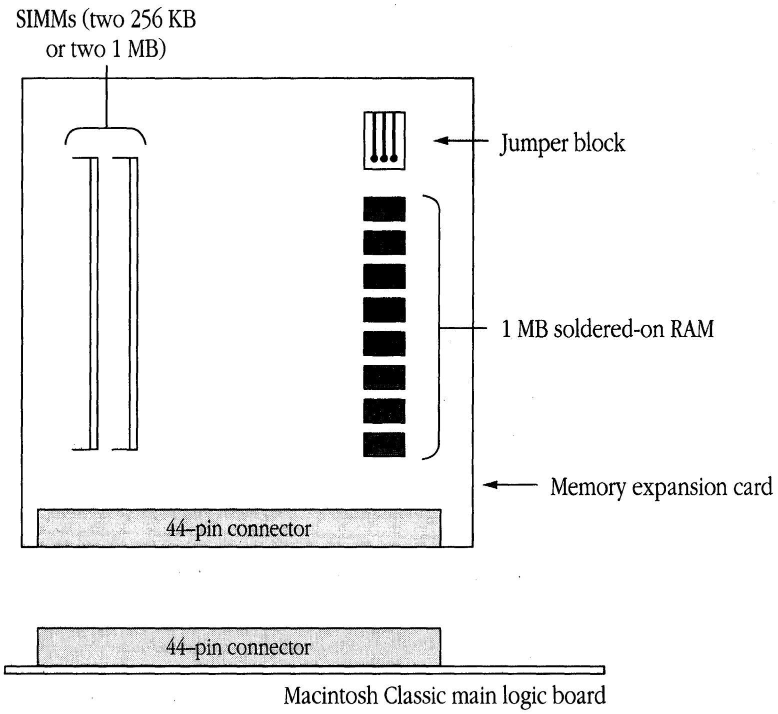


Dimensions are in millimeters with inches in parentheses.

Apple Computer has developed its own unique memory expansion card for the Macintosh Classic. Figure 20-7 provides details of the Apple memory expansion card's configuration. The standard configuration of this memory expansion card consists of 1 MB of additional RAM, a 44-pin connector that mates with the connector on the main logic board, and two SIMM connectors. The 1 MB of additional memory is provided by a bank of eight 256K × 4 DRAMs soldered onto the expansion card.

Installing 256 KB SIMMs in the two SIMM connectors increases total memory to 2.5 MB; installing 1 MB SIMMs increases total memory to 4 MB. A jumper block on the expansion card is used to indicate whether SIMMs are installed.

■ **Figure 20-7** Macintosh Classic RAM expansion card configuration



RAM expansion for the PowerBook 140 and PowerBook 170

This section provides information about RAM expansion cards for the Macintosh PowerBook 140 and Macintosh PowerBook 170 portable computers. The PowerBook 140 and PowerBook 170 computers each come equipped with 2 MB of RAM soldered to the main logic board. Because of the increasing size of application programs, the PowerBook 140 and PowerBook 170 computers are designed to accommodate an expansion card that will provide 2, 4, or 6 MB of additional RAM for the system.

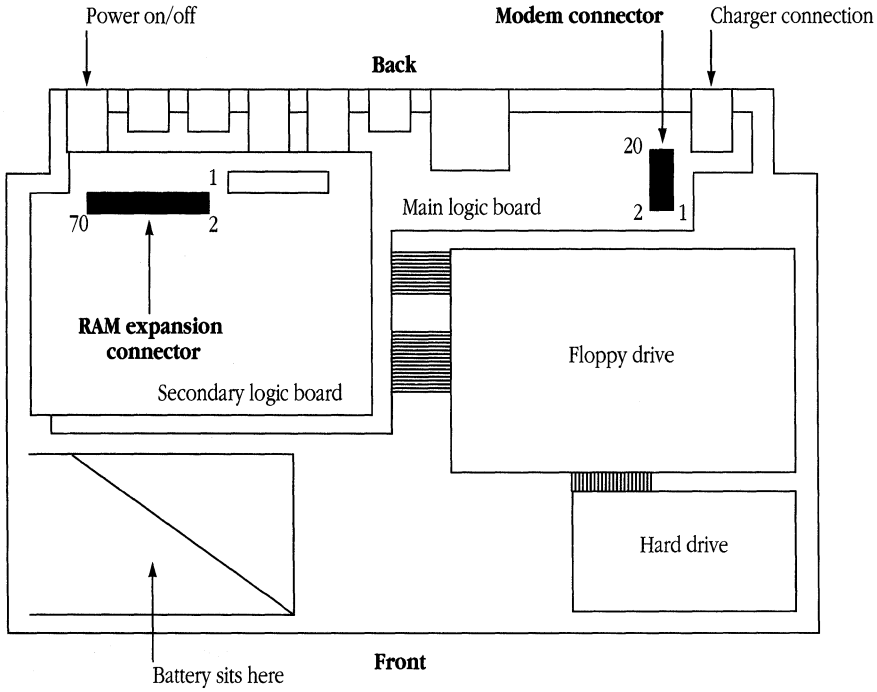
The RAM expansion interfaces for the PowerBook 140 and PowerBook 170 portable computers are identical. A RAM expansion card designed for either of these computers should work correctly in the other one, and also in the Macintosh PowerBook 100.

Topics in this section include a description of RAM expansion connector signals for the PowerBook 140 and the PowerBook 170 and a physical design guide for the RAM expansion cards.

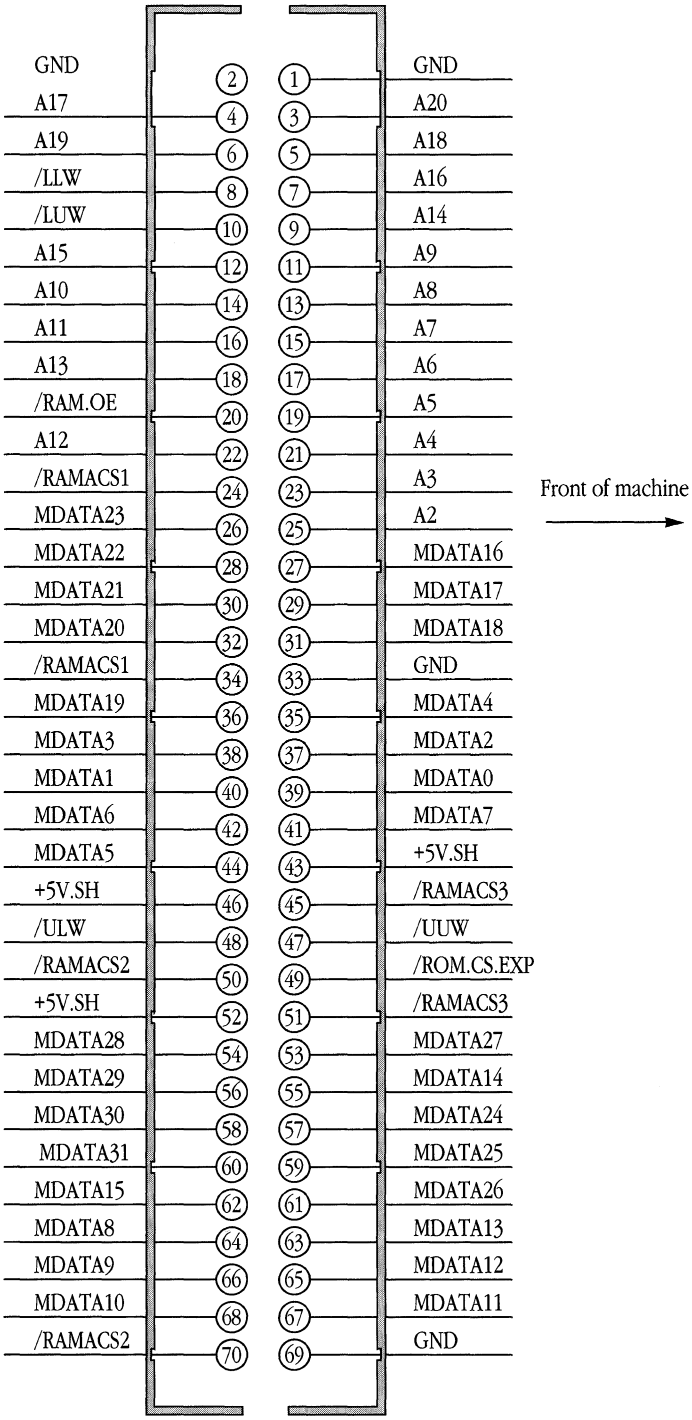
Expansion connector signals for the PowerBook 140 and PowerBook 170

Your RAM expansion card connects to the PowerBook 140 and PowerBook 170 computers through a single 70-pin connector on the secondary logic board. Figure 20-8 shows the physical location and pin orientation of the RAM expansion connector for the PowerBook 140 and PowerBook 170 computers. Figure 20-9 shows the pinout for the RAM expansion connector. Table 20-4 provides the pin number, name, and description of each of the RAM expansion connector signals for the PowerBook 140 and PowerBook 170 computers.

- **Figure 20-8** Location and pin orientation of modem and RAM expansion connectors on the PowerBook 140 and PowerBook 170 computers



■ **Figure 20-9** RAM expansion connector pinout for the PowerBook 140 and PowerBook 170 computers



■ **Table 20-4** RAM expansion connector signals for the PowerBook 140 and PowerBook 170 computers

Pin number	Signal name	Signal description
1	GND	Ground
2	GND	Ground
3	A20	Address bit 20 (unbuffered)
4	A17	Address bit 17 (unbuffered)
5	A18	Address bit 18 (unbuffered)
6	A19	Address bit 19 (unbuffered)
7	A16	Address bit 16 (unbuffered)
8	/LLW	Lower write byte
9	A14	Address bit 14 (unbuffered)
10	/LUW	Lower middle write byte
11	A9	Address bit 9 (unbuffered)
12	A15	Address bit 15 (unbuffered)
13	A8	Address bit 8 (unbuffered)
14	A10	Address bit 10 (unbuffered)
15	A7	Address bit 7 (unbuffered)
16	A11	Address bit 11 (unbuffered)
17	A6	Address bit 6 (unbuffered)
18	A13	Address bit 13 (unbuffered)
19	A5	Address bit 5 (unbuffered)
20	/RAM.OE	RAM output enable and refresh for 4 MB PSRAMs
21	A4	Address bit 4 (unbuffered)
22	A12	Address bit 12 (unbuffered)
23	A3	Address bit 3 (unbuffered)
24	/RAMACS1	PSRAM bank chip-select bit 1
25	A2	Address bit 2 (unbuffered)
26	MDATA23	Bit 23, 32-bit-wide memory data bus (buffered)
27	MDATA16	Bit 16, 32-bit-wide memory data bus (buffered)
28	MDATA22	Bit 22, 32-bit-wide memory data bus (buffered)

(continued)

■ **Table 20-4** RAM expansion connector signals for the PowerBook 140 and PowerBook 170 computers (continued)

Pin number	Signal name	Signal description
29	MDATA17	Bit 17, 32-bit-wide memory data bus (buffered
30	MDATA21	Bit 21, 32-bit-wide memory data bus (buffered)
31	MDATA18	Bit 18, 32-bit-wide memory data bus (buffered)
32	MDATA20	Bit 20, 32-bit-wide memory data bus (buffered)
33	GND	Ground
34	/RAMACS1	PSRAM bank chip-select bit 1
35	MDATA4	Bit 4, 32-bit-wide memory data bus (buffered)
36	MDATA19	Bit 19, 32-bit-wide memory data bus (buffered)
37	MDATA2	Bit 2, 32-bit-wide memory data bus (buffered)
38	MDATA3	Bit 3, 32-bit-wide memory data bus (buffered)
39	MDATA0	Bit 0, 32-bit-wide memory data bus (buffered)
40	MDATA1	Bit 1, 32-bit-wide memory data bus (buffered)
41	MDATA7	Bit 7, 32-bit-wide memory data bus (buffered)
42	MDATA6	Bit 6, 32-bit-wide memory data bus (buffered)
43	+5V.SH	+5 volts (RAM power/shutdown plane)
44	MDATA5	Bit 5, 32-bit-wide memory data bus (buffere)
45	/RAMACS3	PSRAM bank chip-select bit 3
46	+5V.SH	+5 volts (RAM power/shutdown plane)
47	/UUW	Upper write byte
48	/ULW	Upper middle write byte
49	/ROM.CS.EXP	ROM chip select
50	/RAMACS2	PSRAM bank chip-select bit 2
51	/RAMACS3	PSRAM bank chip-select bit 3
52	+5V.SH	+5 volt/0 volt (awake/sleep plane)
53	MDATA27	Bit 27, 32-bit-wide memory data bus (buffered)
54	MDATA28	Bit 28, 32-bit-wide memory data bus (buffered)
55	MDATA14	Bit 14, 32-bit-wide memory data bus (buffered)
56	MDATA29	Bit 29, 32-bit-wide memory data bus (buffered)

(continued)

■ **Table 20-4** RAM expansion connector signals for the PowerBook 140 and PowerBook 170 computers (continued)

Pin number	Signal name	Signal description
57	MDATA24	Bit 24, 32-bit-wide memory data bus (buffered)
58	MDATA30	Bit 30, 32-bit-wide memory data bus (buffered)
59	MDATA25	Bit 25, 32-bit-wide memory data bus (buffered)
60	MDATA31	Bit 31, 32-bit-wide memory data bus (buffered)
61	MDATA26	Bit 26, 32-bit-wide memory data bus (buffered)
62	MDATA15	Bit 15, 32-bit-wide memory data bus (buffered)
63	MDATA13	Bit 13, 32-bit-wide memory data bus (buffered)
64	MDATA8	Bit 8, 32-bit-wide memory data bus (buffered)
65	MDATA12	Bit 12, 32-bit-wide memory data bus (buffered)
66	MDATA9	Bit 9, 32-bit-wide memory data bus (buffered)
67	MDATA11	Bit 11, 32-bit-wide memory data bus (buffered)
68	MDATA10	Bit 10, 32-bit-wide memory data bus (buffered)
69	GND	Ground
70	/RAMACS2	PSRAM bank chip-select bit 2

◆ *Note:* If you are designing a RAM expansion card, you should normally consider pin 49 (/ROM.CS.EXP) as no connection, unless your expansion card includes its own ROM and it is intended to replace system ROM.

If you design a RAM expansion card correctly, it will also work in the PowerBook 100 computer, a 68HC000-based portable computer. The PowerBook 100 RAM expansion interface is discussed later in this chapter. The 68030-based PowerBook 140 and PowerBook 170 computers have a 32-bit data bus, whereas the PowerBook 100 has only a 16-bit data bus. You should design the expansion card as a 32-bit device; but by correctly partitioning the data lines and chip-select lines on the card, you can use the same card in either machine without loss of performance. The card should have 32 data lines coming out to its connector. The chip-select lines for the upper 16 data bits and the lower 16 data bits should be separated to allow for individual selection of either the upper 16 bits or the lower 16 bits of data. The separated chip-select lines are necessary for the 68HC000-based machine because it can only get access to 16 bits at a time. A 68030-based machine does not require separated chip-select lines, because it has a 32-bit data bus. Therefore, the lines are tied back together on the main logic boards of the PowerBook 140 and the PowerBook 170.

As is the case with permanent RAM, only 4 Mbit chips are used for expansion RAM. For example, a 4 MB RAM expansion card has eight 4 Mbit PSRAMs (512K × 8-bit chips arranged as two banks of 32 bits), and a 6 MB card has twelve 4 Mbit PSRAMs (512K × 8-bit chips arranged as three banks of 32 bits). Access and cycle times for these devices are 100 ns.

Access to the RAM (both permanent and expansion RAM) from the main processor requires two processor wait states (five clock cycles per RAM access). Unlike the SRAM in the Macintosh Portable, the PSRAM in the PowerBook 140 and PowerBook 170 computers must be refreshed. The CPU GLU custom chip includes the necessary circuitry to perform the refresh function.

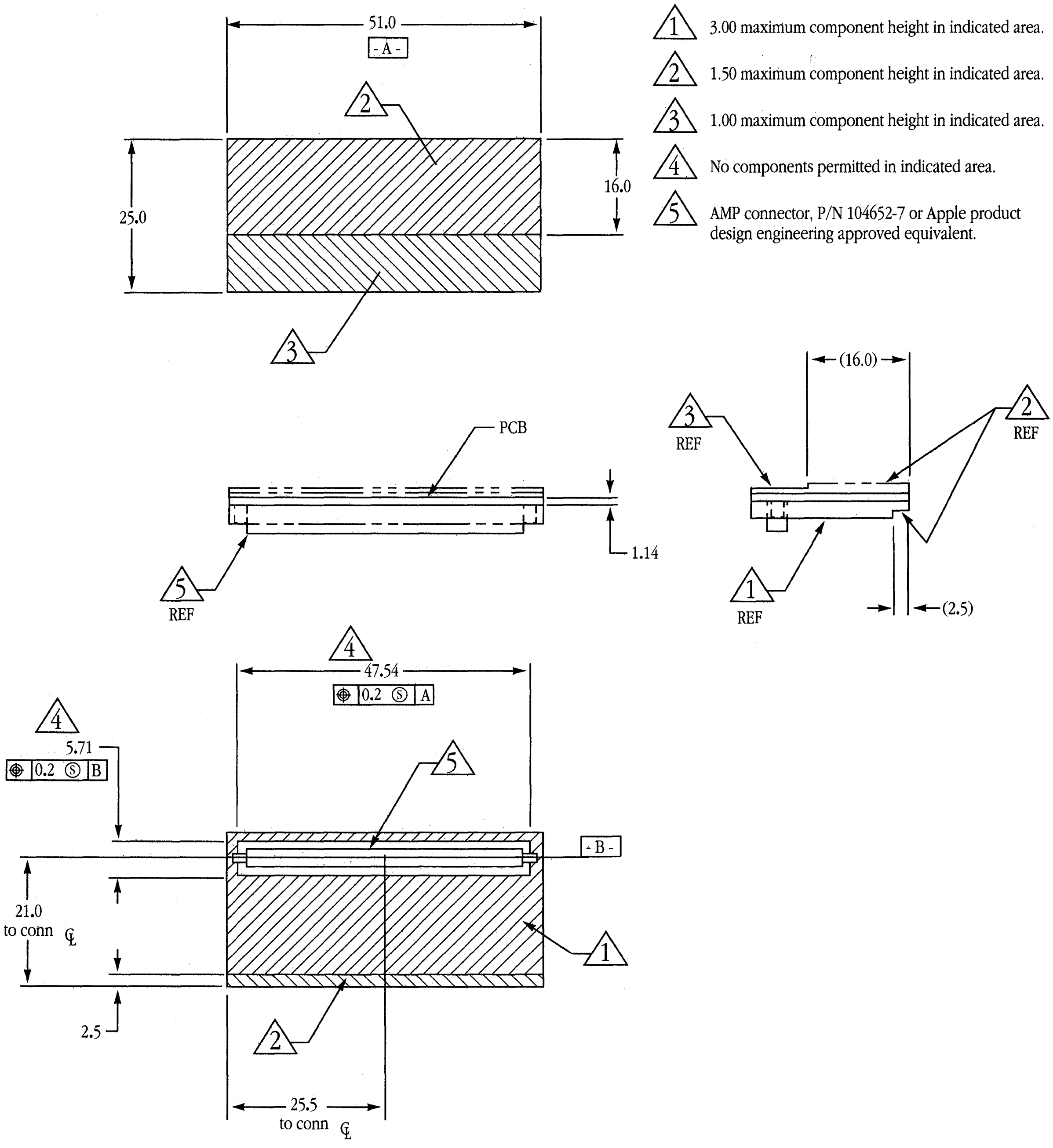
△ **Important** If you are designing a RAM expansion card for the PowerBook 140 or PowerBook 170, you don't have to include logic for address decode or chip select, because all of the required signals (address, data, chip select, and control) are available at the RAM expansion connector. Data buffering is also provided by the PowerBook 140 and PowerBook 170 computers to compensate for the extra loading caused by the RAM expansion card chips. △

RAM expansion card design guide for the PowerBook 140 and PowerBook 170

Figure 20-10 is a design guide providing the physical information you will need to design a RAM expansion card for the PowerBook 140 and PowerBook 170 computers.

◆ *Note:* Figure 20-10 is also applicable to the PowerBook 100 RAM expansion, described in the next section.

- **Figure 20-10** RAM expansion card design guide for the PowerBook 140 and PowerBook 170 computers



1. 3.00 maximum component height in indicated area.
2. 1.50 maximum component height in indicated area.
3. 1.00 maximum component height in indicated area.
4. No components permitted in indicated area.
5. AMP connector, P/N 104652-7 or Apple product design engineering approved equivalent.

Dimensions are in millimeters.

Macintosh PowerBook 100 RAM expansion

The Macintosh PowerBook 100 computer is equipped with 2 MB of PSRAM, soldered to the main logic board. The design of the PowerBook 100 allows you to develop an expansion card that will provide up to 6 MB of additional RAM.

This section describes the electrical and mechanical information you will need to design a RAM expansion card for the PowerBook 100 computer. It also provides some design hints for the expansion card.

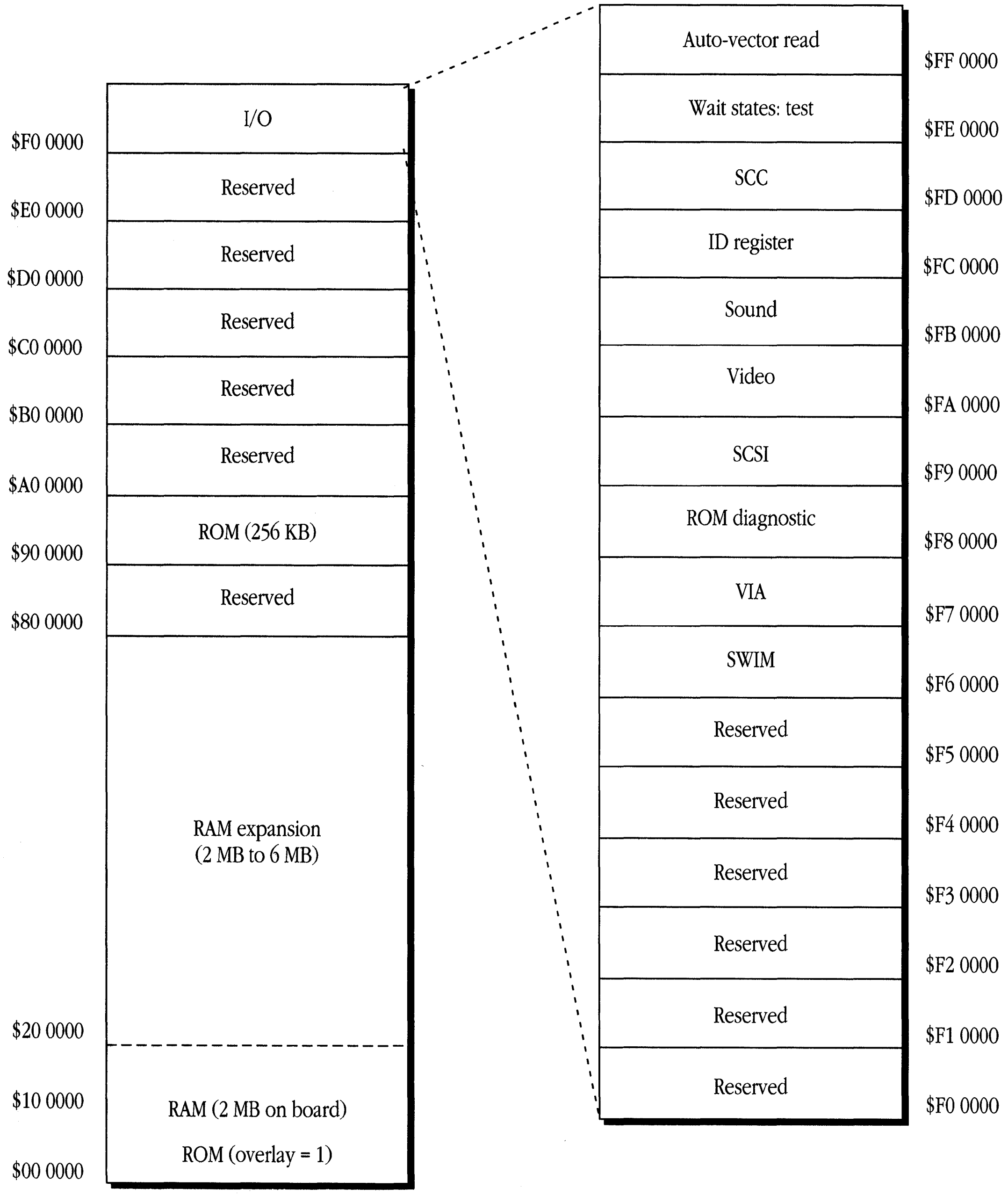
RAM address space for the PowerBook 100

The expansion RAM for the PowerBook 100 computer is located immediately above the permanent RAM in the system memory map. The Miscellaneous GLU determines the amount of permanent memory and maps the expansion RAM to the appropriate location. The expansion RAM is mapped to occupy addresses from $20 0000 through $7F FFFF. For example, a 2 MB expansion card would occupy address space $20 0000 through $3F FFFF, a 4 MB RAM card would occupy address space from $20 0000 through $5F FFFF, and a 6 MB RAM card would occupy address space from $20 0000 through $7F FFFF. The memory expansion address space is always available and, unlike the permanent memory, is not affected by the state of the overlay bit.

Software always sees both the permanent RAM and the expansion RAM in continuous locations on the bottom of the system memory map. There is no wraparound in the memory, and a /DTACK (data transfer acknowledge) signal is generated for any accesses within the address range of $00 0000 through $7F FFFF. You can get access to these address spaces, where there is no RAM; and although a bus error will not occur, the data would be meaningless.

Figure 20-11 shows the memory map for the PowerBook 100 computer.

■ **Figure 20-11** PowerBook 100 memory map

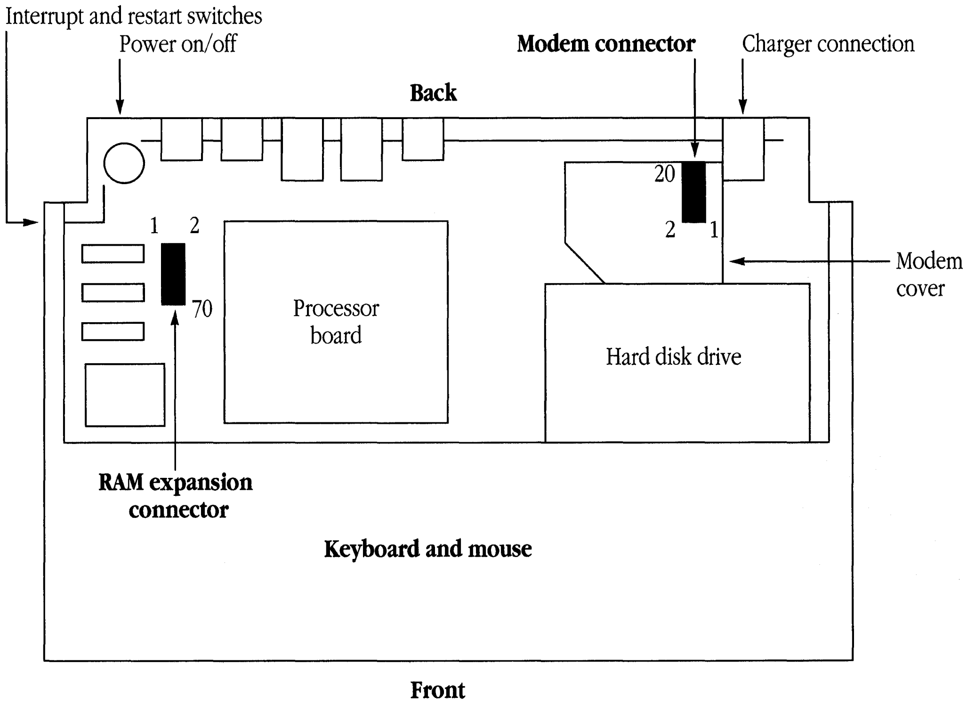


PowerBook 100 RAM expansion connector signals

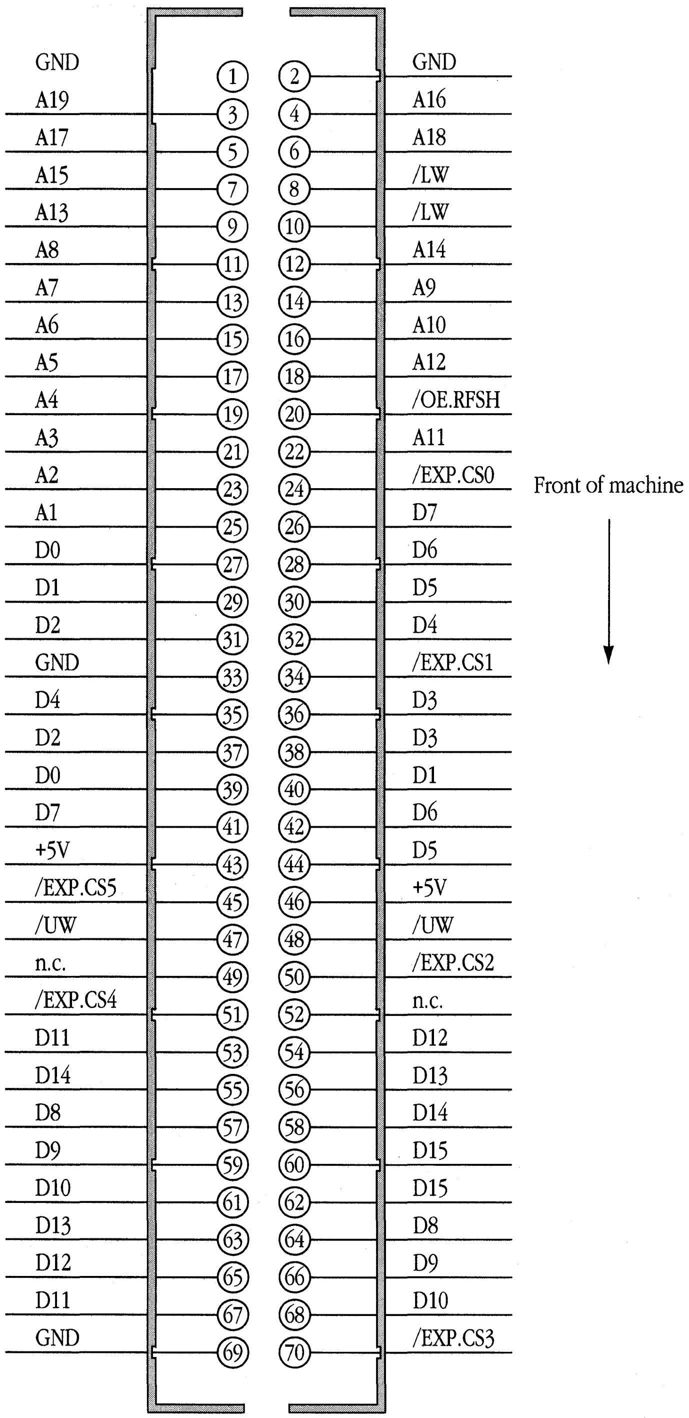
The RAM expansion card connects to the PowerBook 100 through a single 70-pin connector on the main logic board. Figure 20-12 shows the physical location and pin orientation for the RAM expansion connector on the PowerBook 100. Figure 20-13 shows the pinout for the RAM expansion connector.

Table 20-5 provides the pin number, name, and description of each of the RAM expansion connector signals on the PowerBook 100 computer.

■ **Figure 20-12** Location and pin orientation of modem and RAM expansion connectors on the PowerBook 100 computer



■ **Figure 20-13** RAM expansion connector pinout for the PowerBook 100 computer



■ **Table 20-5** PowerBook 100 RAM expansion connector signals

Pin number	Signal name	Signal description
1	GND	Ground
2	GND	Ground
3	A19	Address bit 19 (buffered)
4	A16	Address bit 16 (buffered)
5	A17	Address bit 17 (buffered)
6	A18	Address bit 18 (buffered)
7	A15	Address bit 15 (buffered)
8	/LW	Lower byte write strobe
9	A13	Address bit 13 (buffered)
10	/LW	Lower byte write strobe
11	A8	Address bit 8 (buffered)
12	A14	Address bit 14 (buffered)
13	A7	Address bit 7 (buffered)
14	A9	Address bit 9 (buffered)
15	A6	Address bit 6 (buffered)
16	A10	Address bit 10 (buffered)
17	A5	Address bit 5 (buffered)
18	A12	Address bit 12 (buffered)
19	A4	Address bit 4 (buffered)
20	/OE.RFSH	RAM output enable and refresh
21	A3	Address bit 3 (buffered)
22	A11	Address bit 11 (buffered)
23	A2	Address bit 2 (buffered)
24	/EXP.CS0	Chip-select bit 0
25	A1	Address bit 1 (buffered)
26	D7	Data bit 7 (buffered) to and from main logic board
27	D0	Data bit 0 (buffered) to and from main logic board
28	D6	Data bit 6 (buffered) to and from main logic board
29	D1	Data bit 1 (buffered) to and from main logic board
30	D5	Data bit 5 (buffered) to and from main logic board
31	D2	Data bit 2 (buffered) to and from main logic board

(continued)

■ **Table 20-5** PowerBook 100 RAM expansion connector signals (continued)

Pin number	Signal name	Signal description
32	D4	Data bit 4 (buffered) to and from main logic board
33	GND	Ground
34	/EXP.CS1	Chip-select bit 1
35	D4	Data bit 4 (buffered) to and from main logic board
36	D3	Data bit 3 (buffered) to and from main logic board
37	D2	Data bit 2 (buffered) to and from main logic board
38	D3	Data bit 3 (buffered) to and from main logic board
39	D0	Data bit 0 (buffered) to and from main logic board
40	D1	Data bit 1 (buffered) to and from main logic board
41	D7	Data bit 7 (buffered) to and from main logic board
42	D6	Data bit 6 (buffered) to and from main logic board
43	+5V	+5 volts RAM power
44	D5	Data bit 5 (buffered) to and from main logic board
45	/EXP.CS5	Chip-select bit 5
46	+5V	+5 volts RAM power
47	/UW	Upper byte write strobe
48	/UW	Upper byte write strobe
49	n.c.	Not connected
50	/EXP.CS2	Chip-select bit 2
51	/EXP.CS4	Chip-select bit 4
52	n.c.	Not connected
53	D11	Data bit 11 (buffered) to and from main logic board
54	D12	Data bit 12 (buffered) to and from main logic board
55	D14	Data bit 14 (buffered) to and from main logic board
56	D13	Data bit 13 (buffered) to and from main logic board
57	D8	Data bit 8 (buffered) to and from main logic board
58	D14	Data bit 14 (buffered) to and from main logic board
59	D9	Data bit 9 (buffered) to and from main logic board
60	D15	Data bit 15 (buffered) to and from main logic board
61	D10	Data bit 10 (buffered) to and from main logic board
62	D15	Data bit 15 (buffered) to and from main logic board
63	D13	Data bit 13 (buffered) to and from main logic board

(continued)

■ **Table 20-5** PowerBook 100 RAM expansion connector signals (continued)

Pin number	Signal name	Signal description
64	D8	Data bit 8 (buffered) to and from main logic board
65	D12	Data bit 12 (buffered) to and from main logic board
66	D9	Data bit 9 (buffered) to and from main logic board
67	D11	Data bit 11 (buffered) to and from main logic board
68	D10	Data bit 10 (buffered) to and from main logic board
69	GND	Ground
70	/EXP.CS3	Chip-select bit 3

Design hints for the PowerBook 100 RAM expansion card

The PowerBook 100 computer supports RAM expansion card sizes of 1 to 6 MB in 1 MB increments. The main logic board does not have to be modified to change the RAM configuration when an expansion card is installed.

If you design your RAM expansion card correctly, it will also work in the PowerBook 140 and the PowerBook 170, which are both 68030-based portable computers. The PowerBook 140 and the PowerBook 170 computers are discussed earlier in this chapter. The 68030-based machines have a 32-bit data bus, whereas the 68HC000 in the PowerBook 100 has only a 16-bit data bus. You should design the expansion card as a 32-bit device; but by correctly partitioning the data lines and chip-select lines on the card, you can use the same card in all three machines without loss of performance. The card should have 32 data lines coming out to its connector, and the chip-select lines for the upper 16 data bits and the lower 16 data bits should be separated to allow for individual selection of either the upper 16 bits or the lower 16 bits of data. The separated chip-select lines are necessary for the PowerBook 100 because it can only get access to 16 bits at a time. The 68030-based machines do not require separated chip-select lines because they have a 32-bit data bus; therefore, the lines are tied back together on the computer's main logic board.

As is the case with permanent RAM, only 4 Mbit chips are used for expansion RAM. For example, a 2 MB RAM expansion card has four 4 Mbit PSRAMs (512K × 8-bit chips), a 4 MB RAM expansion card has eight 4 Mbit PSRAMs (512K × 8-bit chips), and a 6 MB card has twelve 4 Mbit PSRAMs (512K × 8-bit chips). Access and cycle times for these devices are 100 ns.

Access to the RAM (both permanent and expansion RAM) from the main processor of the PowerBook 100 requires no (zero) processor wait states, except during the 15 µs refresh cycle. In the PowerBook 100 computer, the Miscellaneous GLU includes special circuitry that performs the refresh function. During sleep and shutdown modes, in which the main processor is powered off and there is no bus access, the PSRAM enters a self-refresh mode to save power.

If you are designing a RAM expansion card for the PowerBook 100, you do not have to include logic for address decode or the chip-select signal, because all of the required signals (address, data, chip select, and control) are available at the RAM expansion connector. The PowerBook 100 also provides address buffering and data buffering to compensate for the extra loading caused by the RAM expansion card chips.

Like permanent main memory for the PowerBook 100, there are no processor wait states when accessing memory locations in internal expansion RAM, with one exception. When PSRAM is being refreshed once every 15 µs, there will be a wait state. During sleep and shutdown, when the processor is powered off and there is no bus access, the PSRAM is placed into self-refresh mode to save power. In the self-refresh mode, no refresh cycle is generated, and refresh is done internal to the RAM chips.

PowerBook 100 RAM expansion card design guide

The RAM expansion card design guide for the PowerBook 100 computer is identical to the design guide for the PowerBook 140 and PowerBook 170 RAM expansion card. Refer to Figure 20-10 for the physical information you will need to design a RAM expansion card for the PowerBook 100 computer.

Chapter 21 **ROM Expansion Interface**

This chapter describes the ROM expansion interface provided in the Macintosh Portable and Macintosh Classic II computers. It describes the electrical characteristics of each ROM expansion connector and the physical specifications for installing a ROM expansion card in the Macintosh Portable and the Macintosh Classic II computers. This chapter also includes a discussion of electronic disks (EDisks) in the Macintosh Portable.

Macintosh Portable ROM expansion

The Macintosh Portable computer is equipped with 256 KB of permanent ROM. The design of the machine allows you to develop an expansion card that will provide up to 4 MB of additional ROM for the system.

This section describes the ROM expansion address space, defines the design criteria, and provides the electrical and mechanical information you need to design a ROM expansion card. It also explains the driver software requirements and provides details for implementing EDisks.

ROM expansion address space in the Macintosh Portable

The 256 KB of processor ROM in the Macintosh Portable is fundamentally similar to the ROM in the Macintosh SE. This ROM is located at the low end of the 1 MB RAM space described in Chapter 20.

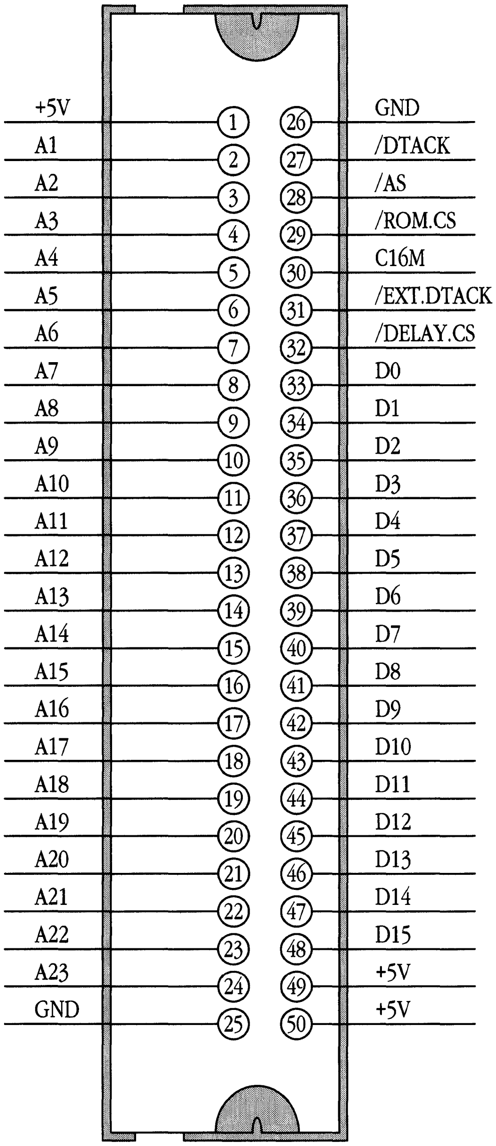
The 1 MB ROM space at locations $90 0000 through $9F FFFF is reserved by Apple primarily as an upgrade path for future ROM code. The 4 MB ROM space at locations $A0 0000 through $DF FFFF is available for your ROM expansion cards.

△ **Important** Although you could design a ROM expansion card to override the existing 1 MB ROM space, it is strongly recommended that you do not, because the machine using that card would be incompatible with many software products. △

ROM expansion cards for the Macintosh Portable

Your ROM expansion card connects to the Macintosh Portable through a single 50-pin connector (slot) on the Macintosh Portable main logic board. Chapter 17 provides information about the location of the connectors on the main logic board. Figure 21-1 shows the pinout for the ROM expansion connector.

■ **Figure 21-1** Macintosh Portable ROM expansion connector pinout



All necessary address bus, data bus, and control signals from the Macintosh Portable are provided to the card through this ROM expansion connector. Table 21-1 provides names and descriptions of each signal. When the expansion card receives these signals, they are decoded into address selects and routed to address and data buffers. Buffering is important to reduce capacitive coupling.

When you design your ROM expansion card, you must remember to include circuitry for decoding, control, and buffering of the signals available at the expansion connector. You should also use CMOS devices, since the maximum current allotted to the ROM expansion connector is only 25 mA. Also, remember that the /DTACK (data transfer acknowledge) signal generated by your card controls the number of wait states.

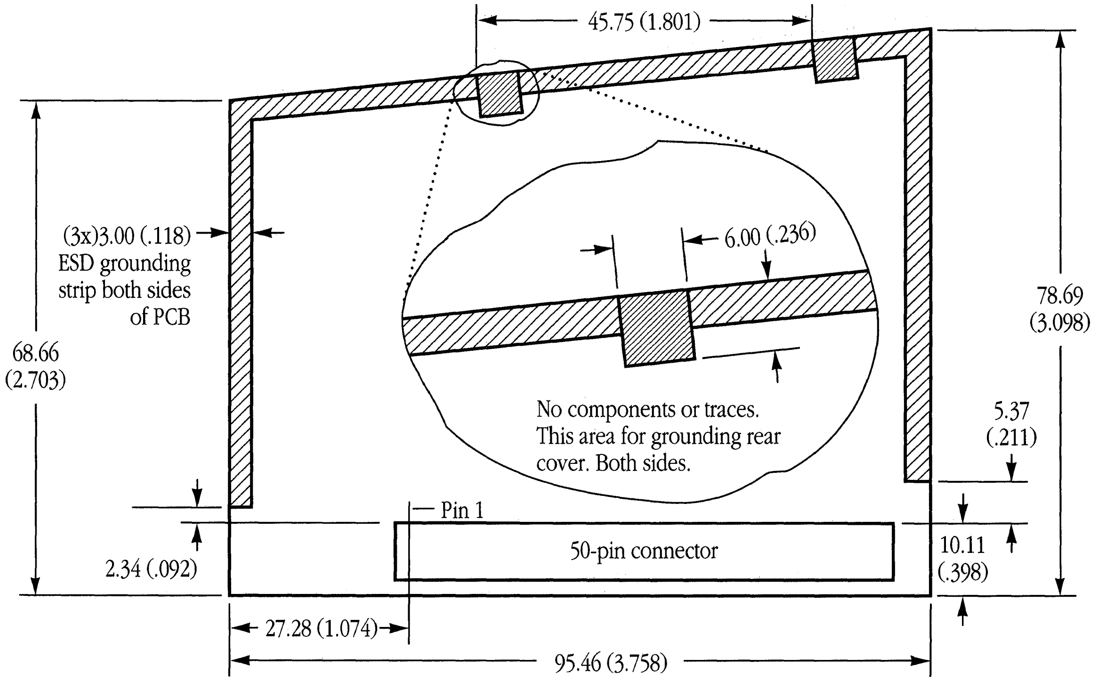
■ **Table 21-1** Macintosh Portable ROM expansion connector signals

Pin number	Signal name	Signal description
1	+5V	+5-volt power supply
2–24	A1–A23	Unbuffered 68HC000 address signals A1–A23
25–26	GND	Logic ground
27	/DTACK	Data transfer acknowledge input to 68HC000
28	/AS	68HC000 address strobe signal
29	/ROM.CS	Permanent ROM chip-select signals; select range from $90 0000 through $9F FFFF
30	C16M	16 MHz system clock
31	/EXT.DTACK	Extended data transfer acknowledge signal that delays generation of the /DTACK signal
32	/DELAY.CS	Signal generated by the CPU GLU chip to put the ROM expansion card into the idle mode by inserting multiple wait states
33–48	D0–D15	Unbuffered 68HC000 data signals D0–D15
49–50	+5V	+5-volt power supply

Figure 21-2 is a design guide providing the physical specifications you need to design a ROM expansion card for the Macintosh Portable.

▲ **Warning** Figure 21-2 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change. ▲

■ **Figure 21-2** Macintosh Portable ROM expansion card design guide



Dimensions are in millimeters with inches in parentheses.

Design considerations for ROM expansion in the Macintosh Portable

In the future, Apple may upgrade the ROM in the Macintosh Portable. Apple will probably do this by producing a ROM expansion card that can override the hard-wired ROM code. The Apple ROM expansion card will have the following characteristics. One side will contain four 32-pin ROM sockets that are compatible with 128K × 8-bit or 512K × 8-bit ROMs, a DIP (dual in-line package) switch for selecting 128 KB or 512 KB addressing sizes for the ROM sockets, and appropriate decoupling capacitors. The other side will have Apple expansion ROMs and any additional circuitry that is necessary.

If, in the meantime, you have already designed your own expansion card with hard-wired ROM code, it would not be compatible with the Apple upgrade, and the user would have to choose between the Apple upgrade and your expansion card.

You can avoid this problem by including standard 32-pin DIP socketed ROMs in your expansion card design. Then if Apple produces a ROM upgrade expansion card, the user can simply transfer the ROMs from your card to the empty sockets on the Apple ROM expansion card. The empty ROM sockets on the Apple ROM expansion card allow you to use either 512 KB or 2 MB of the 4 MB ROM space that is available.

By today's standards, the amount of address space provided for ROM in the Macintosh Portable is large. Even so, the amount of space and the number of ROM chips on a ROM expansion card are limited. When designing your ROM expansion card, therefore, use only the space you really need and, if possible, leave room (address space and empty chip sockets) to add other ROMs. This gives your customers more flexibility by allowing them to insert other developers' ROMs in your expansion card rather than forgoing your card for another design that offers them this flexibility.

Finally, you should make sure your ROM is relocatable. Just because your code is in ROM does not mean it will always reside at a specific address. If your ROM has to be moved to another card (an Apple upgrade or another third-party expansion card), there should be no worry about which socket to place the ROM in or if your address range will conflict with that of another product. Also, ROM expansion may be implemented in some future product with expanded or different address space. Keeping your ROM relocatable could be the difference between additional sales or having an incompatible card that requires an expensive upgrade.

Macintosh Portable EDisks (electronic disks)

You may wish to design your expansion card to function as one or more **EDisks (electronic disks)** that appear to the user to be very fast, silent disk drives. EDisks use RAM or ROM as their storage media, unlike floppy or hard disk drives, which record data on rotating magnetically coated disks.

◆ *Note:* Third-party developers are currently limited to developing only ROM EDisks, not RAM EDisks.

The 4 MB address space allocated for ROM expansion can support a number of ROM EDisks. They must start on a 64 KB boundary, but their size can exceed 64 KB. A ROM EDisk behaves just like an internal RAM EDisk except that it is read-only and cannot be resized.

The EDisk driver for the Macintosh Portable

The EDisk driver provides a system interface to EDisks similar to the Sony and SCSI disk drivers. It supports 512-byte block I/O operations and creates a drive queue element for each EDisk drive but does not support file system tags. It is a ROM `'DRVR'` resource with an `ID` of 48, a `refnum` of –49, and a driver name of `.EDisk`. For information on the driver calls, refer to the disk driver information in *Inside Macintosh.*

The rest of this section describes some of the implementation details, data formats, and algorithms used by the EDisk driver that may be helpful to you if you are designing a ROM expansion card for EDisks.

Data checksumming

To provide better data integrity, the EDisk driver supports checksumming of each data block. The checksum is computed during every write operation to the data block and checked during every read operation. For example, a 32-bit checksum is computed for each 512-byte block by adding each longword in the block to a running longword checksum that is initially 0, but is rotated left by 1 bit before each longword is added in. The following assembly code example demonstrates the algorithm.

```
        Lea         TheBlock,A0        ; A0 is a pointer to the block
                                       ; to checksum
        Moveq.L     #0,D0              ; D0 is the checksum,
                                       ; initially zero
        Moveq.L     #(512/4)-1,D1      ; loop counter for 1 block
                                       ; (4 bytes per iteration)
@Loop   Rol.L       #1,D0              ; rotate the checksum
        Add.L       (A0)+,D0           ; add data to the running
                                       ; checksum
        Dbra        D1,@Loop           ; loop through each longword
                                       ; in the block
```

EDisk driver operation

When the EDisk driver is opened, it searches the address range from the base of the system ROM to $00F0 0000 for ROM EDisks. A ROM EDisk must begin with a valid EDisk header block. (The header block must start on a 64 KB boundary but may be any size.) If a valid header block is found, it is compared with all other headers that have been found, and if it is identical to any one of them it will be ignored, thus eliminating duplicates caused by address wraparound. If the valid header block is unique, a drive queue entry is created for it, and the EDisk driver will now support it. The number of ROM EDisks that can be supported by the driver is limited only by the address space allocated for ROM.

EDisk header format

Associated with each ROM EDisk is a 512-byte header block that describes the layout of the EDisk and uniquely identifies it. The EDisk header marks the beginning of an EDisk. The header should occur at the beginning of the ROM space that is used for EDisk storage (for example, starting at the first byte of a 64 KB ROM block).

The following assembly-language code example gives the general format of the header block. The fields used in the header block are defined following the code example.

```
EDiskHeader        Record 0,increment       ; layout of EDisk signature block

HdrScratch         DS.B      128            ; scratch space for R/W testing and
                                            ; vendor info
HdrBlockSize       DS.W      1              ; size of header block (512 bytes for
                                            ; version 1)
HdrVersion         DS.W      1              ; header version number (this is
                                            ; version 1)
HdrSignature       DS.B      12             ; 45 44 69 73 6B 20 47 61 72 79 20 44
HdrDeviceSize      DS.L      1              ; size of device, in bytes
HdrFormatTime      DS.L      1              ; time when last formatted (pseudo
                                            ; unique ID)
HdrFormatTicks     DS.L      1              ; ticks when last formatted (pseudo
                                            ; unique ID)
HdrCheckSumOff     DS.L      1              ; offset to CheckSum table, if present
HdrDataStartOff    DS.L      1              ; offset to the first byte of data
                                            ; storage
HdrDataEndOff      DS.L      1              ; offset to the last byte 1 of data
                                            ; storage
HdrMediaIconOff    DS.L      1              ; offset to the media icon and mask, if
                                            ; present
HdrDriveIconOff    DS.L      1              ; offset to the drive icon and mask, if
                                            ; present
HdrWhereStrOff     DS.L      1              ; offset to the Get Info Where: string,
                                            ; if present
HdrDriveInfo       DS.L      1              ; longword for Return Drive Info call,
                                            ; if present
                   DS.B      512-*          ; rest of block is reserved
EDiskHeaderSize    EQU       *              ; size of EDisk header block
                   ENDR
```

HdrScratch: The 128-byte `HdrScratch` field is used for read/write testing on RAM EDisks to determine if the memory is ROM or RAM. On ROM EDisks, the vendor should fill in a unique string to identify the version of the ROM EDisk. For example, you might use something like "Copyright 1988, Apple Computer, Inc. System Tools 6.0.3, 12/19/88."

HdrBlockSize: The 2-byte `HdrBlockSize` field indicates the size of the EDisk header block. The size is currently 512 bytes.

HdrVersion: The 2-byte `HdrVersion` field indicates the version of the EDisk. The version number is currently $0001.

HdrSignature: The 12-byte `HdrSignature` field indicates a valid EDisk header block. You must set the signature to these hexadecimal numbers: 45 44 69 73 6B 20 47 61 72 79 20 44.

HdrDeviceSize: The 4-byte `HdrDeviceSize` field indicates the size of the device in bytes. This may be greater than the actual usable storage space. You might also think of the device size as the offset (from the beginning of the header block) of the last byte of the storage device.

HdrFormatTime: The 4-byte `HdrFormatTime` field indicates the time of day when the EDisk was last formatted. The EDisk driver updates this field for RAM-based EDisks when a `Format` control call is made. This information may be useful in uniquely identifying a RAM-based EDisk.

HdrFormatTicks: The 4-byte `HdrFormatTicks` field indicates the value of the system global ticks when the EDisk was last formatted. The EDisk driver updates this field for RAM-based EDisks when a `Format` control call is made. This information may also be useful in uniquely identifying a RAM-based EDisk.

HdrCheckSumOff: The 4-byte `HdrCheckSumOff` field indicates whether checksumming should be performed on the EDisk. This field is set to 0 if checksumming should not be performed on the EDisk. If checksumming is performed, `HdrCheckSumOff` is the offset (from the beginning of the header block) of the checksum table.

HdrDataStartOff: The 4-byte `HdrDataStartOff` field is the offset (from the beginning of the header block) of the first block of EDisk data.

HdrDataEndOff: The 4-byte `HdrDataEndOff` field is the offset (from the beginning of the header block) of the byte after the end of the last block of EDisk data.

HdrMediaIconOff: The 4-byte `HdrMediaIconOff` field is the offset (from the beginning of the header block) of the 128-byte icon and the 128-byte icon mask that represents the disk media. An offset of 0 indicates that the EDisk driver should use the default media icon for this EDisk.

HdrDriveIconOff: The 4-byte `HdrDriveIconOff` field is the offset (from the beginning of the header block) of the 128-byte icon and the 128-byte icon mask that represents the disk drive physical location. An offset of 0 indicates that the EDisk driver should use the default drive icon for this EDisk.

HdrWhereStrOff: The 4-byte `HdrWhereStrOff` field is the offset (from the beginning of the header block) of the Pascal string that describes the disk location for the `FinderGetInfo` command. An offset of 0 indicates that the EDisk driver should use the default string for this EDisk.

HdrDriveInfo: The 4-byte `HdrDriveInfo` field should be returned by the `DriveInfo` control call. A value of 0 indicates that the EDisk driver should use the default drive information for this EDisk.

FPU/ROM expansion for the Macintosh Classic II computer

The Macintosh Classic II computer is equipped with 512 KB of permanent ROM. The design of the machine allows you to develop an expansion card that will provide up to 3 MB of additional ROM for the system.

This section contains the electrical description of the FPU/ROM expansion connector, describes the ROM expansion address space, and provides the mechanical information you need to design an FPU/ROM expansion card for the Macintosh Classic II computer.

Electrical description of the Macintosh Classic II FPU/ROM expansion slot

You can design an FPU/ROM expansion card for the Macintosh Classic II computer that can provide 2 MB or 3 MB of ROM, or a 68882 FPU (floating-point unit) coprocessor, or both. The FPU/ROM expansion card connects to the Macintosh Classic II through a single 50-pin connector (slot) on the main logic board. All necessary signals, including address, data, FPU select, expansion ROM enable, read/write, data strobe acknowledge, reset, and the system clock, are provided to the FPU/ROM slot. Another signal, ROM SELECT, is used in conjunction with a jumper on J9 to control the ROM configuration. Two configurations are possible:

- If the jumper is not connected (default condition) and the ROM SELECT signal is high, you can have 3 MB of additional ROM on the FPU/ROM card but only 1 MB of main ROM socketed to the main logic board.
- If the jumper is connected and the ROM SELECT signal is low, you can have only 2 MB of additional ROM on the FPU/ROM card and 2 MB of main ROM socketed to the main logic board.

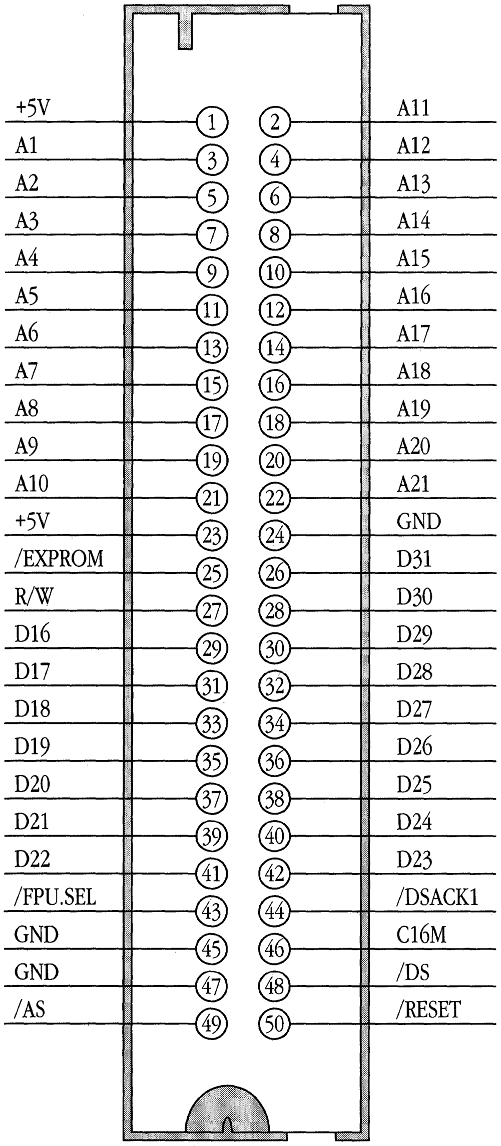
Figure 21-3 shows the pinout for the 50-pin socket connector. Table 21-2 gives the pin number, name, description, and load capacity of each signal supplied to the FPU/ROM expansion connector.

▲ **Warning** Apple strongly discourages the development of a third-party FPU/ROM card for any type of internal expansion other than a 68882 FPU coprocessor or additional ROM. This includes any internal hardware modifications such as clipping on to the main processor. There are several reasons why you should not use the FPU/ROM card for other types of internal development:

- The power budget for the Macintosh Classic II computer allows no margin for additional internal devices.
- The Macintosh Classic II computer's cooling fan cannot dissipate the excess heat that may result.
- Electromagnetic interference (EMI) testing did not take into account the possibility of additional internal devices or the fact that antennae are created whenever additional external cabling is added.
- Any additional load on the Macintosh Classic II computer's data lines could result in noise leading to data errors and unreliable software.
- The address space assigned to the FPU/ROM connector was originally envisioned to support additional ROM; therefore it will support only read access.

The development of an FPU/ROM card for any type of internal expansion other than an FPU or a ROM will not be supported by Apple Macintosh Developer Technical Support (MacDTS) and may void Apple's customer warranty. ▲

■ **Figure 21-3** Macintosh Classic II FPU/ROM expansion connector pinout



■ **Table 21-2** Macintosh Classic II FPU/ROM expansion slot signals

Pin number	Signal name	Description	Load capacity
1	+5V	+5 volts	800 mA
2	A11	Address bit 11	100 µA/8 mA
3	A1	Address bit 1	100 µA/8 mA
4	A12	Address bit 12	100 µA/8 mA
5	A2	Address bit 2	100 µA/8 mA
6	A13	Address bit 13	100 µA/8 mA
7	A3	Address bit 3	100 µA/8 mA
8	A14	Address bit 14	100 µA/8 mA
9	A4	Address bit 4	100 µA/8 mA
10	A15	Address bit 15	100 µA/8 mA
11	A5	Address bit 5	100 µA/8 mA
12	A16	Address bit 16	100 µA/8 mA
13	A6	Address bit 6	100 µA/8 mA
14	A17	Address bit 17	100 µA/8 mA
15	A7	Address bit 7	100 µA/8 mA
16	A18	Address bit 18	100 µA/8 mA
17	A8	Address bit 8	100 µA/8 mA
18	A19	Address bit 19	100 µA/8 mA
19	A9	Address bit 9	100 µA/8 mA
20	A20	Address bit 20	100 µA/8 mA
21	A10	Address bit 10	100 µA/8 mA
22	A21	Address bit 21	100 µA/8 mA
23	+5V	+5 volts	800 mA
24	GND	Ground	800 mA
25	/EXPROM	Expansion ROM enable	100 µA/100 µA
26	D31	Data bit 31	500 µA/1 mA
27	/R/W	Read/write	100 µA/8 mA
28	D30	Data bit 30	500 µA/1 mA
29	D16	Data bit 16	500 µA/1 mA
30	D29	Data bit 29	500 µA/1 mA
31	D17	Data bit 17	500 µA/1 mA

(continued)

■ **Table 21-2** Macintosh Classic II FPU/ROM expansion slot signals (continued)

Pin number	Signal name	Description	Load capacity
32	D28	Data bit 28	500 µA/1 mA
33	D18	Data bit 18	500 µA/1 mA
34	D27	Data bit 27	500 µA/1 mA
35	D19	Data bit 19	500 µA/1 mA
36	D26	Data bit 26	500 µA/1 mA
37	D20	Data bit 20	500 µA/1 mA
38	D25	Data bit 25	500 µA/1 mA
39	D21	Data bit 21	500 µA/1 mA
40	D24	Data bit 24	500 µA/1 mA
41	D22	Data bit 22	500 µA/1 mA
42	D23	Data bit 23	500 µA/1 mA
43	/FPU.SEL	FPU chip select	100 µA/100 µA
44	/DSACK1	Data strobe acknowledge	100 µA/8 mA
45	GND	Ground	800 mA
46	C16M	16 MHz clock from Eagle gate array	100 µA/100 µA
47	GND	Ground	800 mA
48	/DS	Data strobe	100 µA/8 mA
49	/AS	Address strobe	100 µA/8 mA
50	/RESET	System reset	100 µA/8 mA

ROM expansion address space in the Macintosh Classic II computer

The Macintosh Classic II computer's main ROM is implemented as four 32-pin, 128K × 8-bit ICs providing a standard configuration of 512 KB of ROM. By using a configuration of four 256K × 8-bit ICs, you can fill the 1 MB address space that is reserved for main ROM at locations $40A0 0000 through $40AF FFFF. The default condition for the system ROM is 1 MB on the main logic board and 3 MB on an FPU/ROM expansion card. By installing the ROM SELECT jumper on connector J9, you can change the allowable configuration to 2 MB on the main logic board and 2 MB on an FPU/ROM expansion card. The addresses allocated for main and expansion ROM in both the default and jumper conditions are shown in Table 21-3.

■ **Table 21-3** Macintosh Classic II ROM address allocations

Condition	Main ROM	Expansion ROM
Default	$40A0 0000–$40AF FFFF	$40B0 0000–$40DF FFFF
With jumper	$40AF FFFF–$40BF FFFF	$40C0 0000–$40DF FFFF

The address decode and memory-mapping functions in the Macintosh Classic II computer are provided by the Eagle gate array. The Eagle implements two memory address mapping modes: a 24-bit mode and a 32-bit mode. As in other Macintosh computers, a function code control bit determines which map is to be used in the Macintosh Classic II computer. The addressing scheme for the Macintosh Classic II is similar to the one used in the Macintosh LC. Table 21-4 shows the memory map for the Macintosh Classic II computer.

■ **Table 21-4** Macintosh Classic II memory map summary

Function	24-bit mode	32-bit mode
RAM, MB		
2	$00 0000–$1F FFFF	$0000 0000–$001F FFFF
4	$00 0000–$3F FFFF	$0000 0000–$003F FFFF
6	$00 0000–$5F FFFF	$0000 0000–$005F FFFF
10	$00 0000–$9F FFFF	$0000 0000–$009F FFFF
Video main page	$9F 9A80–$9F F000	$009F 9A80–$009F F000
ROM	$A0 0000–$AF FFFF	$40A0 0000–$40AF FFFF
Expansion FPU/ROM	$B0 0000–$DF FFFF	$40B0 0000–$40DF FFFF
I/O space	$F0 0000–$FF FFFF	$50F0 0000–$50FF FFFF

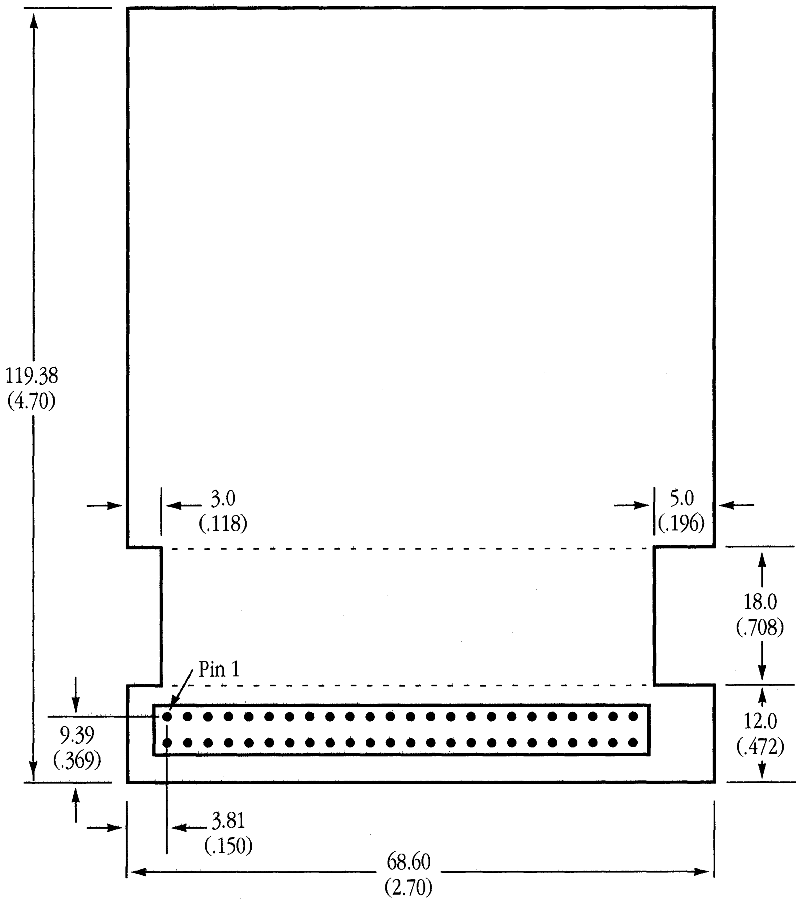
◆ *Note:* Table 21-4 shows the default condition—1 MB of built-in ROM and 3 MB of expansion FPU/ROM slot address space. Refer to Table 21-3 for more details. When the ROM SELECT signal is low (the jumper is installed), ROM addresses are located in 32-bit address space from $40A0 0000 through $40BF FFFF, and expansion FPU/ROM addresses are located in 32-bit address space from $40C0 0000 through $40DF FFFF.

Physical design guidelines for the Macintosh Classic II FPU/ROM expansion card

This section provides the physical information you will need to design an FPU/ROM expansion card for the Macintosh Classic II computer. Figure 21-4 gives the maximum length and width of the expansion card and shows the location of the 50-pin connector.

▲ **Warning** Figure 21-4 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change. ▲

■ **Figure 21-4** Design guide for a Macintosh Classic II FPU/ROM expansion card



Dimensions are in millimeters with inches in parentheses.

Macintosh Classic II FPU/ROM expansion connector power budget

For the FPU/ROM expansion connector in the Macintosh Classic II computer, the +5 VDC voltage that is supplied allows a maximum current load of 800 mA. In addition, power restrictions in the Macintosh Classic II limit the amount of power that can be dissipated by the FPU/ROM expansion card to a maximum of 4 W.

▲ **Warning** Cards dissipating more than 4 W may overheat and damage the circuitry in the Macintosh Classic II computer or cause it to become inoperable. ▲

Chapter 22 **Modem Expansion Interface**

This chapter describes the modem expansion interfaces found in the Macintosh Portable, the Macintosh PowerBook 100, the Macintosh PowerBook 140, and the Macintosh PowerBook 170 computers. This chapter describes the physical and electrical characteristics of the modem cards, the modem hardware interface, the modem power-control interface, and the telephone network interface. It also provides information about communications standards for modems.

Macintosh Portable modem card

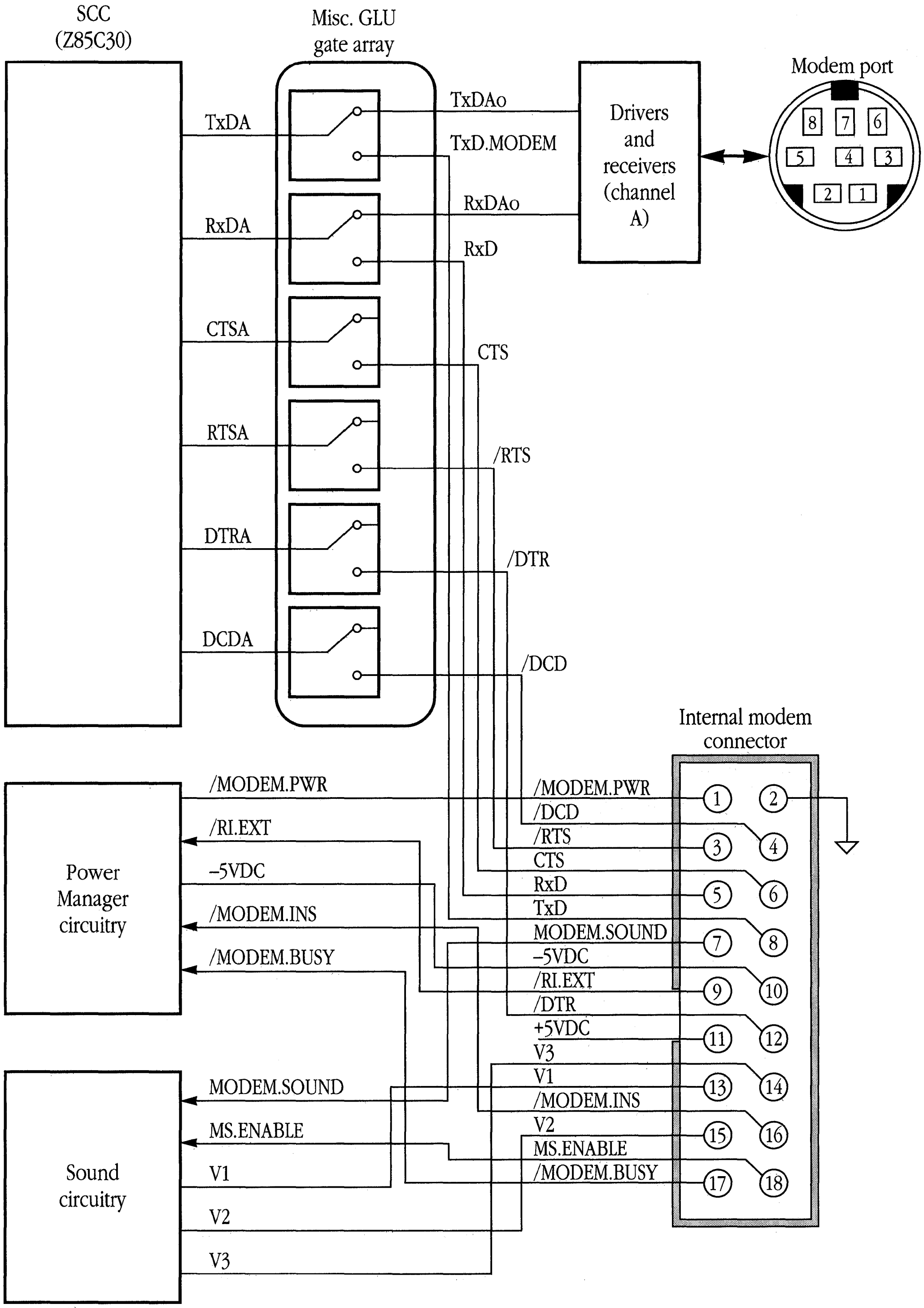
The main logic board of the Macintosh Portable includes an 18-pin, internal modem connector. The connector accommodates an Apple modem card or a compatible third-party modem card.

This section provides information you need if you are developing your own modem card and software for the Macintosh Portable computer.

Modem card hardware interface for the Macintosh Portable

Figure 22-1 shows the hardware interface between a card installed in the modem connector and the Macintosh Portable. Notice that when a compatible modem card is inserted in the modem connector, the card is automatically connected to channel A, the modem port. Although the Macintosh Portable hardware is designed to support operation of the internal modem through either of the two external RS-422 serial ports (printer or modem), the firmware supports operation only through the modem port.

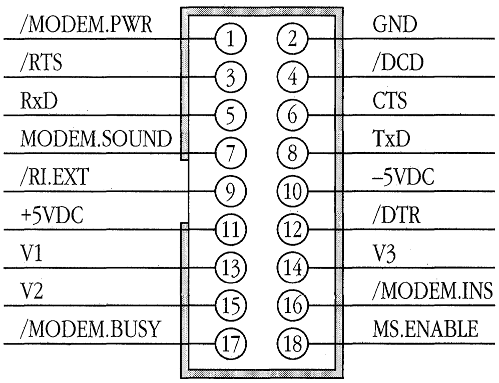
■ **Figure 22-1** Modem interface for the Macintosh Portable



Modem connector electrical interface for the Macintosh Portable

The modem card connects to the Macintosh Portable through an 18-pin dual in-line socket connector (slot). The data is at CMOS levels (V_{IL} = 0 to 0.8 V; V_{IH} = 3.5 to V+; I_{OL} = 1.6 mA; I_{OH} = –25 µA). Figure 22-2 shows the pinout of the modem connector for the Macintosh Portable. Table 22-1 provides the name and description of each signal available at the modem connector. Chapter 17 provides more information about the location of the modem connector on the main logic board.

■ **Figure 22-2** Pinout of modem connector on the Macintosh Portable



■ **Table 22-1** Modem connector signal descriptions

Pin number	Signal name	Signal direction	Signal description
1	/MODEM.PWR	Output	Active-low signal from the Power Manager; see the section "Modem Power-Control Interface for the Macintosh Portable" later in this chapter.
2	GND	—	Ground.
3	/RTS	Output	Request to send signal from the computer to the modem.
4	/DCD	Input	Data carrier detect; the behavior of the /DCD signal depends on the state of the &C command.
5	RxD	Input	Data received; connected to the RxD pin on the SCC.

(continued)

■ **Table 22-1** Modem connector signal descriptions (continued)

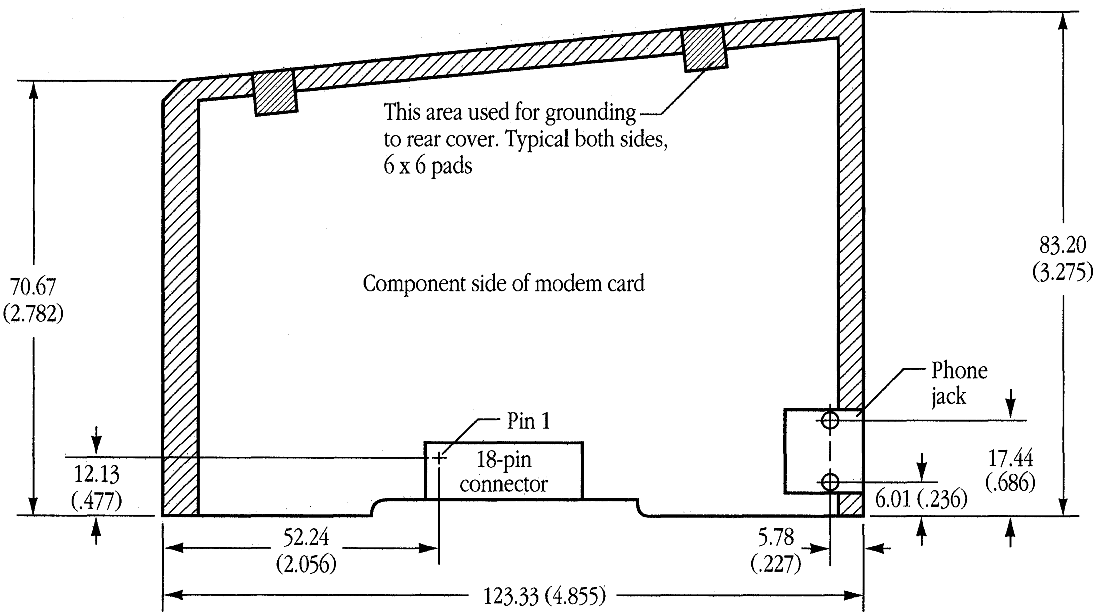
Pin number	Signal name	Signal direction	Signal description
6	CTS	Input	Clear to send; asserted by the modem whenever it has power.
7	MODEM.SOUND	Input	Analog sound; output from the modem.
8	TxD	Output	Transmit data; data and commands from the TxD pin on the SCC.
9	/RI.EXT	Input	Ring detect interrupt; the signal to the computer that a ring is present. If the computer is in the sleep state, assertion of this signal causes the computer to return to the operating state and power up the modem.
10	–5VDC†	—	–5 V power; the –5 V supply is guaranteed to be present whenever the /MODEM.PWR signal is asserted. This signal may float or go to ground any time following the negation of /MODEM.PWR.
11	+5VDC	—	VCC power; whenever the host has power available, this pin supplies +5.2 VDC ±5%.
12	/DTR	Output	Data terminal ready.
13	V1	Output	Least significant volume-control bit. This signal may remain high following the negation of /MODEM.PWR.
14	V3	Output	Most significant volume-control bit. This signal may remain high following the negation of /MODEM.PWR.
15	V2	Output	Second volume-control bit. This signal may remain high following the negation of /MODEM.PWR.
16	/MODEM.INS	Input	Modem installed; always asserted by the modem while the modem is installed in the computer.
17	/MODEM.BUSY	Input	Modem busy; asserted by the modem whenever the modem is busy.
18	MS.ENABLE	Input	Modem sound enable; asserted by the modem to enable the computer's speaker.

† Power on pin 10 is controlled by the Power Manager IC.

Physical design guide for a Macintosh Portable modem card

Figure 22-3 provides the mechanical information you will need to design a modem card for the Macintosh Portable, including the maximum dimensions and the location of the 18-pin connector.

- **Figure 22-3** Modem card design guide for the Macintosh Portable



Dimensions are in millimeters with inches in parentheses.

Modem power-control interface for the Macintosh Portable

Two signal lines, /MODEM.PWR and /MODEM.BUSY, control power to the modem connector. A modem card can use the /MODEM.BUSY signal to indicate to the CPU that any of the following is true:

- The modem is executing its power-up sequence.
- The modem is off-hook (for any reason).
- The modem is executing a command (if Hayes compatible).

If the modem is executing any self-tests, it is considered to be executing a command and therefore busy.

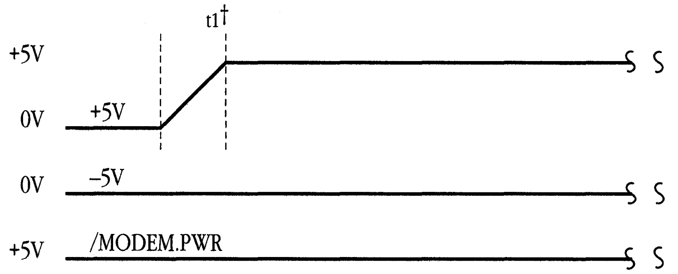
Power-up/power-down timing

The Macintosh Portable includes a Power Manager IC that controls the /MODEM.PWR signal. See Figures 22-4 and 22-5 for timing diagrams of the cold-start (initial power-up) and warm-start (wake-up) power sequences. If /MODEM.PWR is negated (high), the modem must immediately initiate its power-off sequence regardless of what it is doing. The modem must enter the sleep state within 500 ms following the negation of /MODEM.PWR; by that time the modem must reduce its power consumption to meet the maximum power limitation for sleep state. (For more information, see the section "Modem Card Power Requirements" later in this chapter.) The modem can also use that 500 ms to set its outputs to a default state and store its operating parameters and register values so that it can restore them when operation resumes. Two of the lines to the modem, /DTR and TxD, go to ground potential within 50 ns of the negation of /MODEM.PWR. While the computer is in the sleep state, the volume-control bits V1, V2, and V3 are floating.

◆ *Note:* On the Macintosh Portable, the CTS line is always asserted (high) because flow control is not provided. The /RI.EXT signal always reflects the status of the incoming ring signal. The /RTS signal, which is meaningless in full-duplex operation, is not connected. When /MODEM.PWR is negated and the modem card prepares itself for the sleep state, the card forces two of its outputs (/DCD and RxD) high and one of its outputs (MS.ENABLE) low.

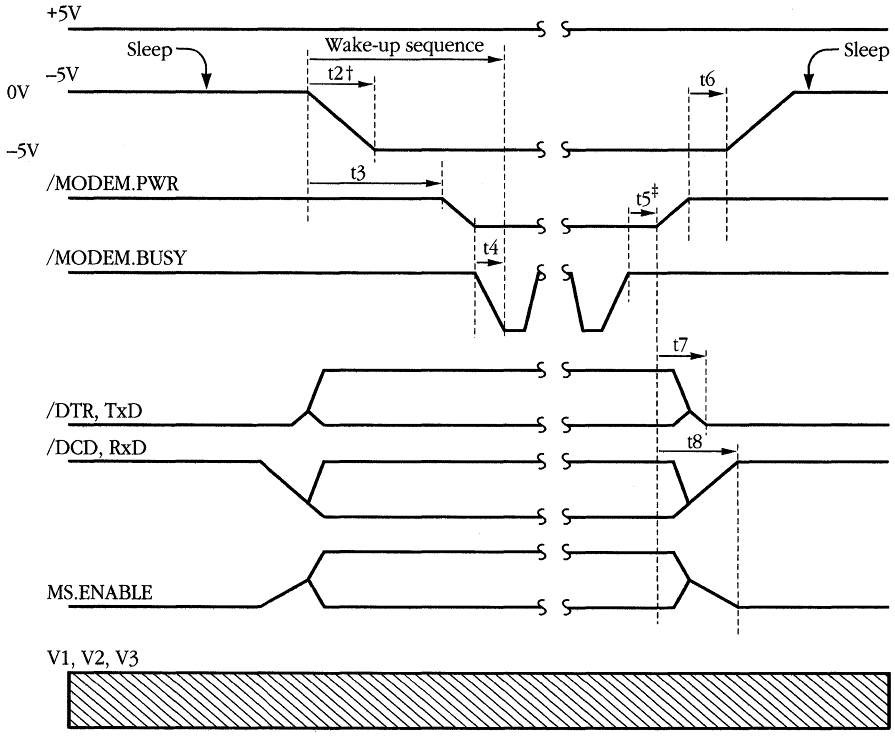
Usually, the Power Manager does not negate /MODEM.PWR if the modem has /MODEM.BUSY asserted. However, there are times when the Power Manager IC must turn the modem off even though it is busy (for example, when the battery reserve is too low). If this occurs, the modem must stop its activity (for example, go on-hook) and perform the necessary activities to prepare for switching to its sleep state. If the modem is executing a command when /MODEM.PWR is negated, the modem can do one of two things before switching to its sleep state: either finish executing the command or abort execution and restore the state prior to the command, whichever takes the least amount of time.

■ **Figure 22-4** Cold-start timing diagram for the Macintosh Portable



† After t1, maximum overshoot is within 50 mV peak to peak.

■ **Figure 22-5** Warm-start timing diagram for the Macintosh Portable



	t1	t2	t3	t4	t5‡	t6	t7	t8
Minimum			100 ms		0	500 ms		
Typical	2 ms	35 ms						
Maximum	30 ms	70 ms		30 ms			50 ns	500 ms

† After t2, maximum overshoot is within 50 mV peak to peak.
‡ The Macintosh Portable may not obey this minimum time.

Ring detection

The ring detect interrupt (/RI.EXT) signal is asserted during most of the AC cycle of a ring and is used to signal the computer that a ring is taking place. Both ringing and pulsing can trigger the ring detector. The microprocessor in your modem should be capable of distinguishing between ring and pulse dialing by detecting the frequency of the incoming signal. If the modem is turned off, the Macintosh Portable can power it up and determine whether the /RI.EXT signal corresponds to a ring or a pulse by reading the appropriate register or looking for the appropriate result code.

Modem card power requirements

A modem card must be able to operate on +5.2 VDC ±5% and –5.0 VDC ±5%. This voltage is supplied through the modem connector by either the Macintosh Portable battery or a combination of battery and charger. Maximum power consumption by the modem card when fully operational is 750 mW; however, a modem card typically consumes 525 mW of power when in the operating state. In the sleep state, power consumption is only 3 mW.

During normal operation, both +5 VDC and –5 VDC are provided to the modem card. During the sleep state, only +5 VDC is supplied.

Telephone network interface

Your modem design may require a balanced, two-wire telephone interface meeting FCC Part 68 and Part 15 Class B and DOC rules.

It should include one 8-wire RJ-11 jack, wired as follows:

- pin 3 for TIP signal
- pin 4 for RING signal

Pins 1, 2, 5, and 6 are not used.

Installing an RJ-11 jack on the rear of the modem card allows a common RJ-11 plug (used on single-line telephone equipment) to be inserted, completing the connection of a telephone to the modem.

Modem expansion cards for the PowerBook-family computers

The main logic boards on the Macintosh PowerBook 100, the Macintosh PowerBook 140, and the Macintosh PowerBook 170 provide a 20-pin, internal modem connector. The connector accommodates an Apple modem card or a compatible third-party modem card.

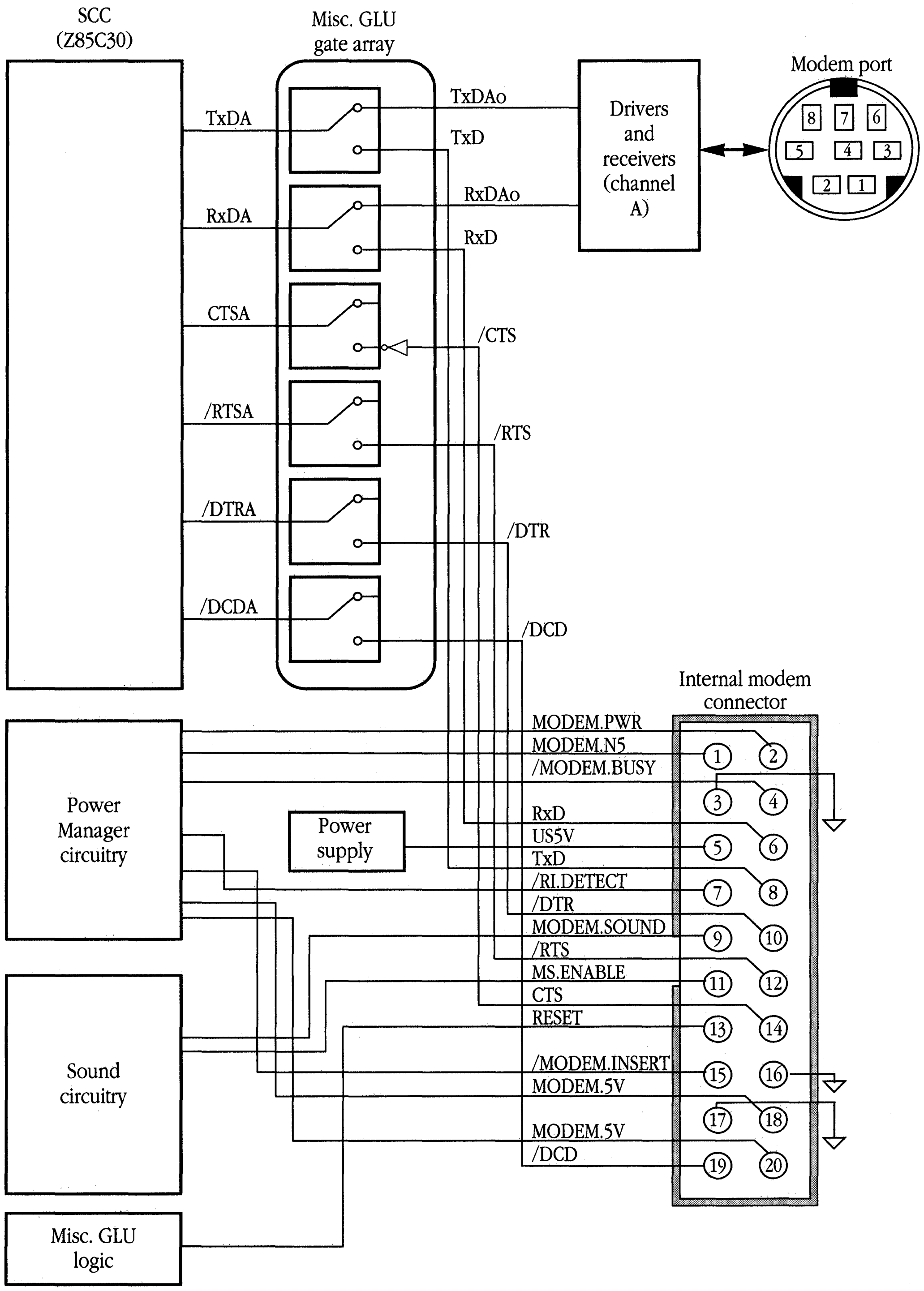
This section provides information that will help you in your development of a modem card and software.

Modem card hardware interface for the PowerBook-family computers

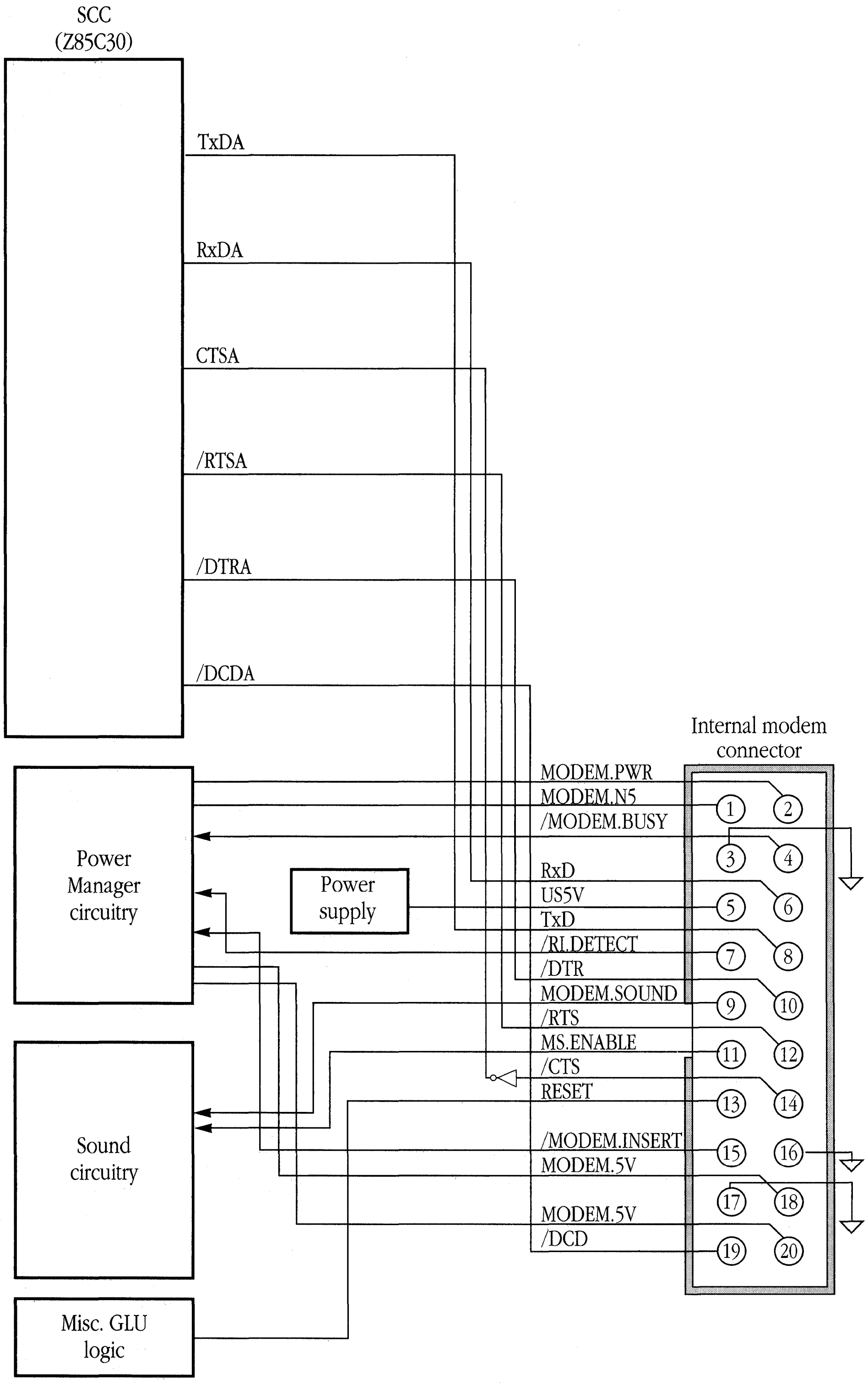
Figure 22-6 shows the hardware interface for a card installed in the modem connector on the PowerBook 140 and PowerBook 170 computers. Notice that when a compatible modem card is inserted in the modem connector on the PowerBook 140 and PowerBook 170, the card is automatically connected to channel A, the modem port. Although the hardware in the PowerBook 140 and PowerBook 170 computers is designed to support operation of the internal modem through either of the two external RS-422 serial ports (printer or modem), the firmware supports operation only through the modem port.

Figure 22-7 shows the modem card hardware interface for the PowerBook 100 computer. The modem connector for the PowerBook 100 is directly connected to channel A of the SCC. The PowerBook 100 does not have an external modem port. Channel B (the printer port) is the only external RS-422 serial port on the PowerBook 100.

■ **Figure 22-6** Modem interface for the PowerBook 140 and PowerBook 170 computers



■ **Figure 22-7** Modem interface for the PowerBook 100 computer



Modem connector electrical interface for the PowerBook-family computers

The modem card connects to the PowerBook-family computers through a 20-pin dual in-line socket connector. The data is at CMOS levels (V_{IL} = 0 to 0.8 V; V_{IH} = 3.5 to V+; I_{OL} = 1.6 mA; I_{OH} = 25 µA).

Figures 22-6 and 22-7 show the pinouts of the modem connector. Table 22-2 provides the pin number, name, type, and description of each signal available at the modem connector for the PowerBook-family computers.

■ **Table 22-2** Modem connector signals for the PowerBook family

Pin	Signal name	Signal type	Signal description
1	MODEM.N5		–5 V power that is controlled by the host and provided to the modem circuitry. This may float or go to ground 500 ms following the negation of MODEM PWR. *This signal is not used by the Apple modem.*
2	MODEM.PWR	Output	Active-high signal, open collector output from the Power Manager; see the section "Modem Power-Control Interface for the PowerBook Family" later in this chapter.
3	GND		Electrical ground.
4	/MODEM.BUSY	Input	Modem busy; active-low signal asserted by the modem and sent to the Power Manager whenever the modem is busy.
5	US5V		+5 V power; provides +5 VDC ±5% to the modem whenever the computer has power available.
6	RxD	Input	Receive data; data received by the modem and then sent to the computer via the RxD pin on the SCC.
7	/RI.DETECT	Input	Ring detect; active-low signal sent to the Power Manager to indicate that a ring is present. If the computer is in a "sleep" mode, the assertion of this signal causes the computer to awake and power up the modem.
8	TxD	Output	Transmit data; data and commands that are sent from the computer to the modem via the TxD pin on the SCC.

(continued)

■ **Table 22-2** Modem connector signals for the PowerBook family (continued)

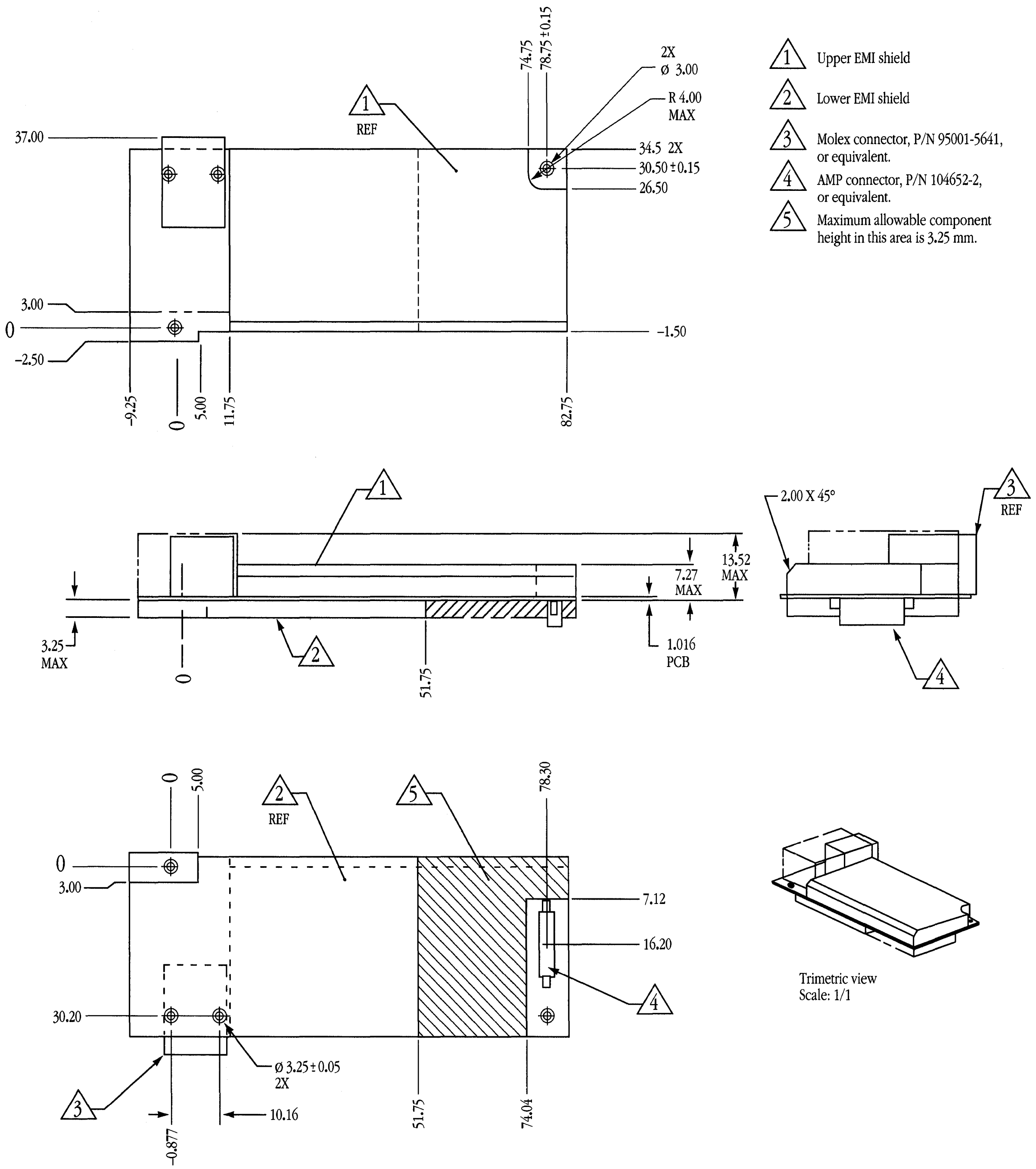
Pin	Signal name	Signal type	Signal description
9	MODEM.SOUND	Input	Modem sound; analog sound high-impedance signal sent by the modem to the computer's sound circuitry.
10	/DTR	Output	Data terminal ready; an active-low signal whose behavior depends on the state of the &D command.
11	MS.ENABLE	Input	Modem sound enable; an active-high signal asserted that the modem sends to the computer's sound circuitry whenever the modem's sound monitor is on.
12	/RTS	Output	Request to send; an active-low signal sent by the computer to the modem via the /RTS pin on the SCC.
13	RESET	Output	Reset; an active-high signal asserted by the Power Manager for a minimal duration and sent to the modem. Reset is asserted after the Power Manager switches –5 V power to the modem or any time the modem needs to be reset.
14	/CTS	Input	Clear to send; an active-low signal asserted by the modem as a default option and sent to the computer via the /CTS pin on the SCC.
15	/MODEM.INSERT	Input	Modem inserted; an active-low signal continuously asserted by the modem, and sent to the Power Manager, whenever the modem card is installed in the computer.
16	GND		Electrical ground.
17	GND		Electrical ground.
18	MODEM.5V		+5 V power that is controlled by the Power Manager and provided to the modem. This pin will float or go to ground 500 ms after the MODEM.PWR signal goes low (inactive).
19	/DCD	Input	Data carrier detect; an active-low signal, driven by the modem, whose behavior depends on the state of the &C command.
20	MODEM.5V		Same as pin 18.

◆ *Note:* If you're building a modem card for the PowerBook 100, the PowerBook 140, or the PowerBook 170, you must place a 10 kΩ pull-up resistor between MODEM.5V and MODEM.PWR.

Physical design guide for the PowerBook-family modem expansion card

Figure 22-8 is a physical design guide giving you the mechanical specifications you will need, including card size and connector location, to design a compatible modem card for the PowerBook 100, PowerBook 140, and PowerBook 170 computers. For the physical location and pin orientation of the modem connector, refer to Figure 20-8 for the PowerBook 140 and PowerBook 170 computers or Figure 20-12 for the PowerBook 100 computer.

■ **Figure 22-8** Modem card design guide for the PowerBook family



1 Upper EMI shield

2 Lower EMI shield

3 Molex connector, P/N 95001-5641, or equivalent.

4 AMP connector, P/N 104652-2, or equivalent.

5 Maximum allowable component height in this area is 3.25 mm.

Dimensions are in millimeters.

Modem power-control interface for the PowerBook family

Two lines from the computer, US5V and MODEM.5V, provide +5 VDC power to the modem. The US5V line is always present unless there is a hardware shutdown (following a battery failure or if the computer's back-panel switch is turned off). The MODEM.5V power is turned on or off depending on the current power mode of the modem and on how the serial port is used. For example, MODEM.5V is turned off when the computer enters the shutdown or sleep mode and when the serial driver is closed.

The modem has two power modes: power-on and standby. Power-on is the normal operating mode. In standby mode, MODEM.5V is switched off, all modem circuits are turned off, and the only source of power is US5V. This mode has very low power consumption since only leakage current should be drawn.

Two signal lines, MODEM.PWR and /MODEM.BUSY, control power to the modem connector from the Power Manager. The /MODEM.BUSY signal is sent to the Power Manager to prevent the computer from removing power to the modem while the modem is using the communication channel to the computer. A modem card uses the /MODEM.BUSY signal to indicate to the computer that any of the following is true:

- The modem is executing its power-up sequence.
- The modem is off-hook (for any reason).
- The modem is executing a command where command execution begins with a carriage return at the end of an AT command sequence or the repeat last command sequence
("a/" or "A/").

◆ *Note:* If the modem is executing any self-tests, it is considered to be executing a command and is therefore busy.

The Power Manager controls the MODEM.PWR signal. If the Power Manager negates MODEM.PWR (signal goes low), it is designed to wait at least 500 ms before turning off MODEM.5V. This delay gives the modem sufficient time to save the communication parameters in EEPROM before MODEM.5V is switched off. Three of the modem interface signals, /DTR, TxD, and /RTS, go to ground potential within 50 ns of the negation of MODEM.PWR.

Usually, the Power Manager does not negate MODEM.PWR if the modem has /MODEM.BUSY asserted. There are times, however, when the Power Manager must turn the modem off even though it is busy—for example, when the battery reserve voltage becomes too low. If this occurs, the modem stops its busy activity (for example, goes on-hook) and performs the necessary activities for switching to standby. The modem can do one of two things if it is executing a command when MODEM.PWR is negated: either finish executing the command, or abort execution and restore the state prior to the command, whichever takes less time.

Modem operation for the PowerBook family

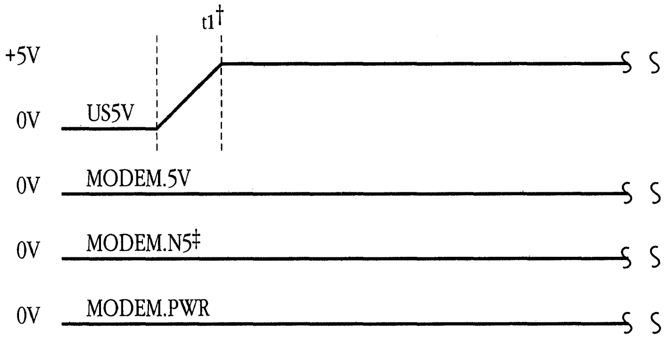
When MODEM.5V power is turned on, the modem leaves standby mode and enters the power-on sequence. A positive RESET signal from the Power Manager resets the modem's microprocessor and begins the initialization sequence, which includes a memory check, the restoration of communications parameters, and the generation of a beep.

If the modem is in standby mode and detects an incoming call (/RI.DETECT is asserted low), the computer acknowledges the call and powers up the modem to check whether the ring is valid. The Power Manager should power up the modem within 5 seconds after /RI.DETECT is asserted.

Power-up/power-down timing

Timing diagrams for the modem's power-up and power-down sequences are shown in Figures 22-9 through 22-11.

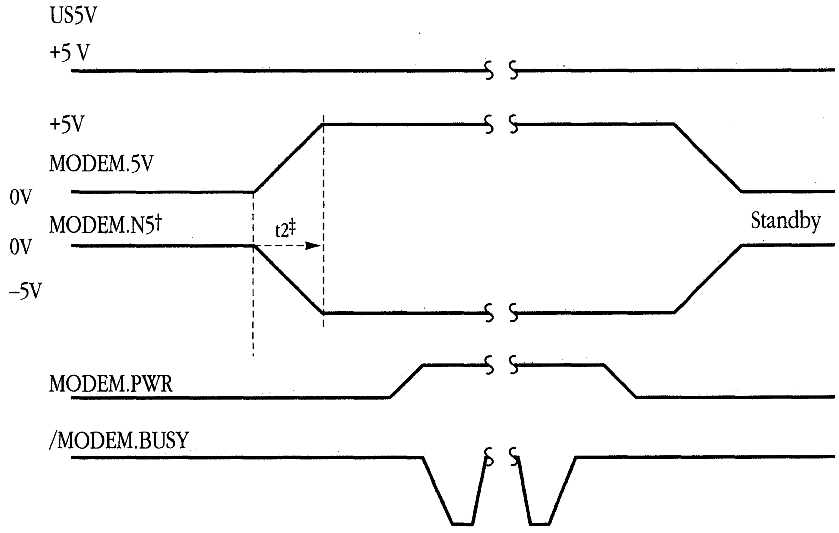
■ **Figure 22-9** Modem cold-start timing diagram for the PowerBook family



† t1 = 2 ms (typical), 30 ms (maximum). After t1, maximum overshoot is less than 50 mV peak to peak.

‡ MODEM.N5, although shown in this diagram, is not used by the Apple modem.

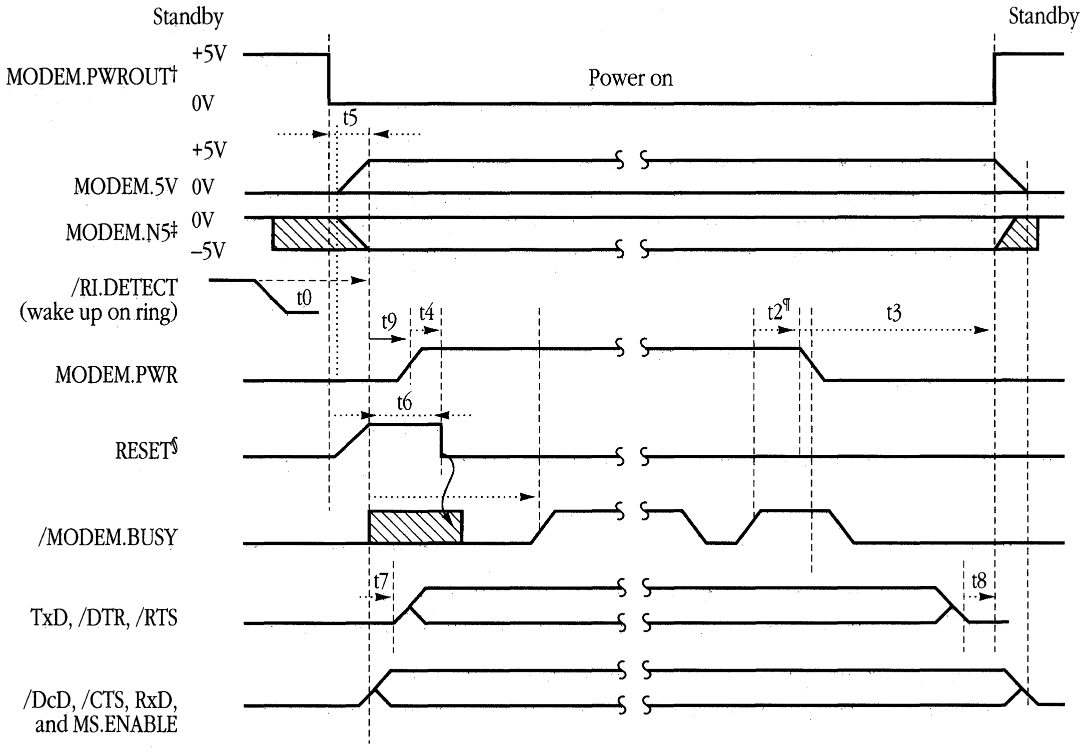
■ **Figure 22-10** Modem warm-start timing diagram for the PowerBook family



† MODEM.N5, although shown in this diagram, is not used by the domestic version of the Apple modem.

‡ t2 = 35 ms (typical), 70 ms (maximum). After t2, maximum overshoot is less than 50 mV peak to peak.

■ **Figure 22-11** Complete power-up/power-down sequence and timing diagram for the PowerBook family



Time:	t0	t2	t3	t4	t5	t6	t7	t8	t9
Minimum	–	0¶	500 ms	0	0	5 ms	0	0	0
Maximum	5 sec	–	–	–	2 ms	–	–	–	–

† MODEM.PWROUT is an internal CPU signal that turns MODEM.5V on and off. US5V is always on and is not shown on this diagram.

‡ MODEM.N5, although shown in this diagram, is not used by the domestic version of the Apple modem.

§ RESET may rise with MODEM.5V, but not before.

¶ t2 > 0 may not be obeyed by the CPU.

Ring detection

The ring detect (/RI.DETECT) interrupt signal is asserted during most of the AC cycle of a ring signal and is used to signal the computer that a ring is taking place. Both ringing and pulsing can trigger the ring detector. The microprocessor in your modem must be capable of distinguishing between ring and pulse dialing by detecting the frequency of the incoming signal. If the modem is turned off, the computer can determine whether the /RI.DETECT signal corresponds to a ring or a pulse by powering up the modem and reading the appropriate register or looking for the RING result code.

Modem card power requirements

A modem card must be able to operate on +5 VDC ±5% and –5 VDC ±5%. These voltages are provided through the modem connector by either the battery or a combination of battery and charger. Typically, a fully operational modem card has an optimized power consumption of 450 mW.

Current drawn by the modem from the two +5 VDC pins on the modem connector should not exceed

- 95 mA typical when in full operation (on line)
- 1 µA when in standby mode and there is no incoming ring signal

Telephone network interface

Your modem design may require a balanced, two-wire telephone interface meeting FCC Part 68 and Part 15 Class B and DOC rules.

It should include one 6-wire RJ-11 type jack wired as follows:

- pin 3 for TIP signal
- pin 4 for RING signal

Pins 1, 2, 5, and 6 are not used.

Installing the RJ-11 type jack on the rear of the modem card allows a common RJ-11 plug (used on single-line telephone equipment) to be inserted, completing the connection of a phone to the modem.

Standards information for reference

The following compilations of signal characteristics are provided for reference only.

Compatibility and modulation

Standard	*Speed (bps)*	*Modulation*	*Baud*
CCITT V.22 bis	2400	QAM	600
CCITT V.22	1200	DPSK	600
CCITT V.21	300/110	FSK	300/110
Bell 212A	1200	DPSK	600
Bell 103	300/110	FSK	300/110

Transmit carrier frequencies

V.22 bis/V.22/212A		*Transmit Carrier*
Originate		1200 Hz
Answer		2400 Hz
Bell 103	*Mark*	*Space*
Originate	1270	1070
Answer	2225	2025
V.21	*Mark*	*Space*
Originate	980	1180
Answer	1650	1850

Guard tone frequencies and transmit levels (CCITT only)

1800 ±20 Hz at 6 ±1 dB below the transmit carrier level
550 ±20 Hz at 3 ±1 dB below the transmit carrier level

Answer tone frequency

V.22 bis/V.22/V.21	2100 Hz
Bell 103/212A	2225 Hz

Received signal frequency tolerance

Offset frequency ±7 Hz

Chapter 23 **Macintosh IIci Cache Memory Expansion**

This chapter provides the electrical and mechanical information you need to design a cache memory expansion card for the Macintosh IIci computer.

Cache memory expansion overview

The Macintosh IIci is equipped with a special-purpose 120-pin Euro-DIN connector designed specifically as a processor-direct interface for a cache memory expansion card. The connector used on the cache card and the mating connector on the main logic board of the Macintosh IIci are physically the same as those used for the Macintosh SE/30 and are shown in Figures 17-22 and 17-23, respectively. The pinout, however, is different. The signals provided in the cache connector are optimized for cache design, not as a general-purpose interface, as is the case with the 68030 Direct Slot connector used on computers such as the Macintosh SE/30, Macintosh IIfx, and Macintosh IIsi.

▲ **Warning** Cards designed for the Macintosh IIci cache connector are not compatible with cards designed for the general-purpose 68030 Direct Slot on other Macintosh computers. Their pinouts, form factors, clock speeds, and power budgets are different. Any attempt to interchange the cards may severely damage both the computer and the card. ▲

If you are determined to design an expansion card other than a cache memory card, you should be aware of the following limitations.

- The cache connector has less power allotted to it than does the 68030 Direct Slot; 5 W of power is allocated at +5V, and +12V is not available.
- The Macintosh IIci case does not have sufficient space for an external device access opening. Therefore you cannot install a connector and cable that would allow your card to have access to external hardware.
- The absence of some machine-specific signals imposes severe restrictions on your design.

△ **Important** Apple strongly recommends that the cache connector be used only for cache memory cards. △

How the cache works

A memory cache is a relatively inexpensive hardware addition that improves CPU performance. It contains a very fast memory, usually SRAM (static random-access memory), that stores data likely to be used on a regular basis. You can think of the cache as a duplicate of a small portion of main memory in that it holds an image of what is in main memory. When the processor searches for a piece of information in main memory, the cache checks to see if it has the information in its data memory, and if it does, the cache immediately provides the information to the processor so that the processor does not have to wait for the slower main memory to provide it.

Both the processor and the cache acquire new data when the main memory places it on the data bus. The entire memory access cycle takes slightly longer, since there is not only the time for a regular memory access but also the time it takes the cache to determine whether or not it has the requested data. Despite this increase in time, the cache design results in a noticeable increase in performance because the data that the processor needs is more often than not in the cache.

Typically, the cache determines if the data it is storing is the same data that the processor requested by comparing the physical addresses that the processor places on the address bus with the addresses stored in the cache tag memory. (The tag memory contains the addresses of the information stored in the cache data memory.) If there is a valid comparison, the information in the cache data memory is sent to the processor.

Using the cache

Your cache expansion card should operate transparently to user programs. The cache is based on physical addresses; it has no access to the logical addresses inside the MC68030 processor, so cache coherency should not be a problem. Also, there is no reason to have to flush the cache unless it is being enabled or a /RESET signal is issued. The 68030 on-chip memory management unit marks the NuBus slot space and all I/O space of the Macintosh IIci as noncacheable, and if the processor addresses them, data in these spaces is not cached.

Your card should not attempt to cache data from accesses made by bus masters other than the MC68030 processor, because other bus masters may not know how to retry. Apple strongly suggests that you use synchronous logic (clocked by the CPUCLK signal) in your cache card design.

Gaining access to the cache card

The 16 MB address space from $5200 0000 through $52FF FFFF in the Macintosh IIci memory map is reserved for cache memory. Table 23-1 shows the 8 MB address spaces that are reserved for the cache data and tag memories.

■ **Table 23-1** Cache memory address space

Cache memory type	From	To
Data	$5200 0000	$527F FFFF
Tag	$5280 0000	$52FF FFFF

Since no select signal is provided on the cache expansion connector, your card design must include appropriate circuitry for decoding the address ranges. The card's address space is not accessible through the 24-bit memory map. Test software running in 24-bit mode must use the `SwapMMUMode` trap to enter the 32-bit mode before it can gain access to the cache card memory.

ROM traps control the enabling, disabling, and flushing of the cache card. These functions are called using a selector from the `HWPriv` (`A098`) trap. See Table 23-2.

■ **Table 23-2** Cache control trap

Function	Selector
`EnableExtCache`	4
`DisableExtCache`	5
`FlushExtCache`	6

The organization of a particular cache card's data and tag memory is determined by the card. System software does not make any assumptions about the card's organization, and only the card's test software should directly address cache card RAM.

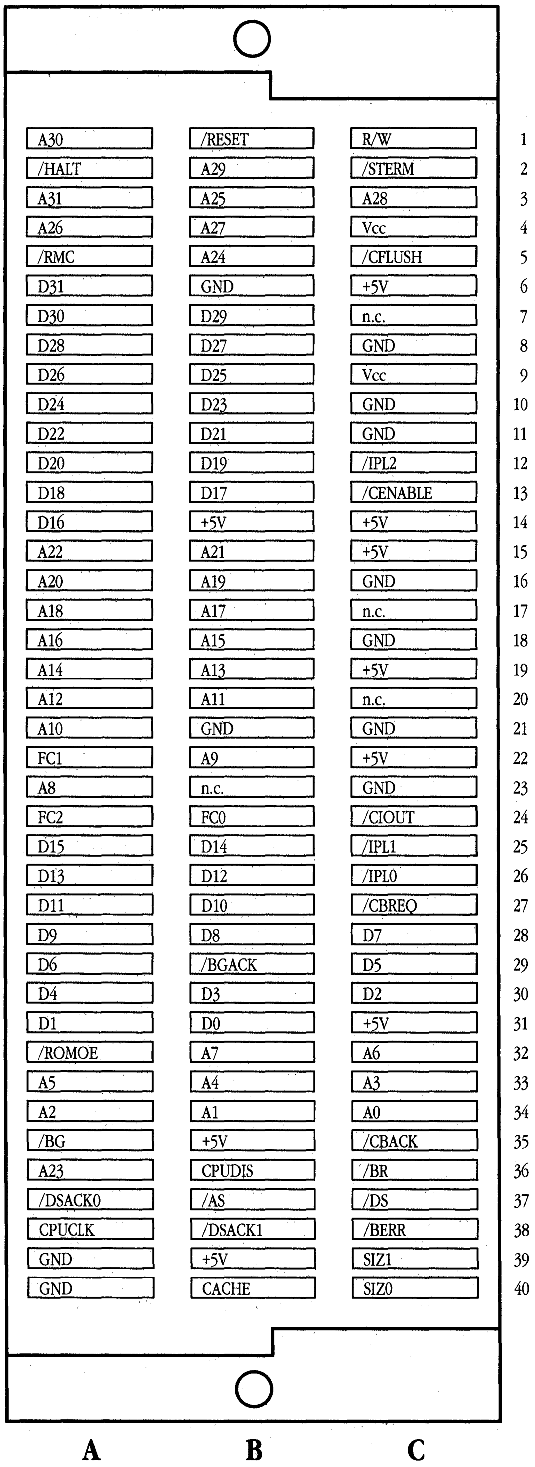
Electrical description of the cache connector

The cache connector is a unique processor-direct slot whose pinout is specifically tailored for cache implementation. Figure 23-1 gives the pinout for the cache connector on the Macintosh IIci main logic board, as viewed from above. In addition to the functional signals required for operation of the cache, the connector provides some special signals for diagnostic testing.

Diagnostic signals (/ROMOE, /DSACK0–/DSACK1, /IPL0–/IPL2, /BR, and CPUDIS) are needed for Apple's internal use in debugging and for emulator support. They are documented so that third-party developers can easily make use of emulators or other hardware debugging tools. The diagnostic signals are not required for cache memory operations and may not be supported in future implementations of the cache connector.

Table 23-3 lists the cache connector signal names and gives a brief description of each signal.

■ **Figure 23-1** Macintosh IIci cache connector pinout



	A	B	C
1	A30	/RESET	R/W
2	/HALT	A29	/STERM
3	A31	A25	A28
4	A26	A27	Vcc
5	/RMC	A24	/CFLUSH
6	D31	GND	+5V
7	D30	D29	n.c.
8	D28	D27	GND
9	D26	D25	Vcc
10	D24	D23	GND
11	D22	D21	GND
12	D20	D19	/IPL2
13	D18	D17	/CENABLE
14	D16	+5V	+5V
15	A22	A21	+5V
16	A20	A19	GND
17	A18	A17	n.c.
18	A16	A15	GND
19	A14	A13	+5V
20	A12	A11	n.c.
21	A10	GND	GND
22	FC1	A9	+5V
23	A8	n.c.	GND
24	FC2	FC0	/CIOUT
25	D15	D14	/IPL1
26	D13	D12	/IPL0
27	D11	D10	/CBREQ
28	D9	D8	D7
29	D6	/BGACK	D5
30	D4	D3	D2
31	D1	D0	+5V
32	/ROMOE	A7	A6
33	A5	A4	A3
34	A2	A1	A0
35	/BG	+5V	/CBACK
36	A23	CPUDIS	/BR
37	/DSACK0	/AS	/DS
38	CPUCLK	/DSACK1	/BERR
39	GND	+5V	SIZ1
40	GND	CACHE	SIZ0

■ **Table 23-3** Macintosh IIci cache connector signal descriptions

Signal name	Signal description
A0–A31	Address bus, bits 0 through 31
D0–D31	Data bus, bits 0 through 31
/AS	Address strobe
/BERR	Bus error
/BG	Bus grant
/BGACK	Bus grant acknowledge
/BR	Bus request
CACHE	Memory controller for cache access
/CBACK	Cache burst acknowledge
/CBREQ	Cache burst request
/CIOUT	Cache inhibit out
/CENABLE	Cache enable
/CFLUSH	Cache flush
CPUCLK	25 MHz CPU clock
CPUDIS	CPU disable
/DS	Data strobe
/DSACK0–/DSACK1	Data transfer and size acknowledge, bits 0 and 1
FC0–FC2	Function codes, bits 0 through 2
GND	Ground
/HALT	Halt
/IPL0–/IPL2	Interrupt priority lines, bits 0 through 2
/RESET	System reset
/RMC	Read-modify-write cycle
/ROMOE	ROM output enable
R/W	Read/write
SIZ0–SIZ1	Transfer size, bits 0 and 1
/STERM	Synchronous termination
n.c.	Not connected
+5V	+5 volts

◆ *Note:* Special-purpose diagnostic signals are shown in boldface.

Table 23-4 indicates whether the signals are inputs or outputs, and provides the load presented or the drive available to each pin of a cache card inserted in the expansion connector.

In the column labeled "Input/Output" in Table 23-4, *In* refers to a signal from the cache card to the processor and corresponds directly to the load presented. *Out* refers to a signal from the processor to the cache card and corresponds directly to the drive available. The last column in Table 23-4, labeled "Load or Drive Limits," gives several specifications. An example may be helpful in interpreting this column. The /BERR signal is shown as presenting a load of 100 μA/8 mA, 50 pF. This is the maximum expected load that the cache card must drive when sending a /BERR signal to the main logic board. The DC load is given in the format *signal high/signal low*. This means that the cache card must drive a load of up to 100 μA when it drives /BERR high (logic 1), and a load of up to 8 mA when it drives /BERR low (logic 0). The AC load is given as 50 pF, the maximum capacitance to ground presented by the main logic board to AC signals (or signal transitions) from the cache card. The notation "1 kΩ pull-up" in the table means that the signal is driven low, and that a 1 kΩ pull-up resistor on the main logic board returns the line to a logic 1.

Correspondingly, /BERR presents a drive of 40 μA/0.4 mA, 30 pF. This is the maximum amount of drive from the main logic board that is available to circuits on a cache memory expansion card. The /BERR signal can drive a cache card DC load of up to 40 μA in the high (logic 1) state, or up to 0.4 mA in the low (logic 0) state. The AC drive is given as 30 pF, the maximum capacitance to ground that a cache expansion card may present to AC signals (or signal transitions) from the /BERR line.

In Table 23-4, where a signal is shown in parentheses, it is usually an output that is driven by the MC68030 but is tristated by the processor after responding to a bus request. When tristated by the MC68030, this signal may be driven by the cache expansion card.

The special-purpose diagnostic signals in Table 23-4 are shown in boldface. In addition to the special diagnostic signals, some of the functions shown in Table 23-4 are used only during diagnostic testing. A dagger following an input or output designation or a load or drive parameter indicates that that particular function is active during diagnostic operations only. For example, under the "Input/Output" column for the /CBACK signal, the word *In* is followed by a dagger. This means that for diagnostic test purposes only, the card can drive a load of up to 100 μA when it drives /CBACK high (logic 1), or a load of up to 8 mA in the low (logic 0) state. Under normal cache operations, the /CBACK input signal is inactive.

■ **Table 23-4** Macintosh IIci cache connector signals, loading or driving limits

Signal name	Input/output	Load or drive limits
A0–A29	(In)†/Out	(Load: 300 µA/1 mA, 100 pF)† Drive: 40 µA/0.4 mA, 30 pF
A30–A31	(In)†/Out	(Load: 300 µA/8 mA, 100 pF)† Drive: 40 µA/0.4 mA, 30 pF 1 kΩ pull-up
D0–D23	In/Out	Load: 150 µA/1 mA, 100 pF Drive: 40 µA/0.4 mA, 30 pF
D24–D31	In/Out	Load: 300 µA/1 mA, 100 pF Drive: 20 µA/0.2 mA, 30 pF
/RESET	In†/Out	Load: na/15 mA, 50 pF† Drive: 20 µA/0.2 mA, 15 pF Open collector, 470 Ω pull-up
/BERR	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF 1 kΩ pull-up
/HALT	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF 1 kΩ pull-up
FC0–FC2	(In)†/Out	(Load: 100 µA/8 mA, 50 pF)† Drive: 20 µA/0.2 mA, 15 pF 1 kΩ pull-up
/BR	In†	Drive: 40 µA/0.4 mA, 30 pF† 1 kΩ pull-up†
/BG	In†/Out	(Load: 100 µA/8 mA, 50 pF)† Drive: 40 µA/0.4 mA, 30 pF 1 kΩ pull-up
/BGACK	Out	Drive: 40 µA/0.4 mA, 30 pF
SIZ0–SIZ1	(In)†/Out	(Load: 40 µA/0.4 mA, 30 pF)† Drive: 40 µA/0.4 mA, 30 pF

(continued)

■ **Table 23-4** Macintosh IIci cache connector signals, loading or driving limits (continued)

Signal name	Input/output	Load or drive limits
/AS	(In)†/Out	(Load: 300 µA/8 mA, 100 pF)† Drive: 40 µA/0.4 A, 30 pF 1 kΩ pull-up
/DSACK0–/DSACK1	In†/Out†	Load: 100 µA/8 mA, 50 pF† Drive: 40 µA/0.4 mA, 30 pF† 1 kΩ pull-up†
R/W	(In)†/Out	(Load: 300 µA/8 mA, 100 pF)† Drive: 40 µA/0.4 A, 30 pF 1 kΩ pull-up
/STERM	In/Out†	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/0.4 mA, 30 pF† 1 kΩ pull-up
/CBACK	In†/Out	Load: 100 µA/8 mA, 50 pF† Drive: 40 µA/0.4 A, 30 pF 1 kΩ pull-up
/CBREQ	In†/Out	Load: 100 µA/8 mA, 50 pF† Drive: 40 µA/0.4 mA, 30 pF 1 kΩ pull-up
/CIOUT	(In)†/Out	(Load: 100µA/8 mA, 50 pF)† Drive: 40 µA/0.4 A, 30 pF 1 kΩ pull-up
/DS	(In)†/Out	(Load: 100 µA/8 mA, 50 pF)† Drive: 40 µA/0.4 mA, 30 pF 1 kΩ pull-up†
/RMC	(In)†/Out†	(Load: 100 µA/8 mA, 50 pF)† Drive: 40 µA/0.4 mA, 30 pF 1 kΩ pull-up†
/IPL0–/IPL2	Out†	Drive: 40 µA/0.4 mA, 30 pF† 1 kΩ pull-up†
CPUCLK	Out	Drive: 10 µA/10 µA, 15 pF

(continued)

■ **Table 23-4** Macintosh IIci cache connector signals, loading or driving limits (continued)

Signal name	Input/output	Load or drive limits
/ROMOE	Out†	Drive: 40 µA/0.4 mA, 30 pF†
CPUDIS	In†	Load: 8 mA/1 mA, 30 pF† 1 kΩ pull-down†
CACHE	In	Drive: 8 mA/1 mA, 30 pF 1 kΩ pull-down
/CFLUSH	Out	Drive: 40 µA/0.4 A, 30 pF
/CENABLE	Out	Drive: 40 µA/0.4 A, 30 pF

† Signal is used for debugging and emulation only.

Electrical design guidelines for the cache card

Most of the cache connector signals are specified to drive two 74LS inputs. (A standard 74LS input load is 20 µA high, 0.2 mA low.) Some other signals, such as /RESET, the high-order data (D24–D31), and the function codes (FC0–FC2), drive only one 74LS input. The CPUCLK signal drives only a CMOS input (a standard CMOS load is 10 µA high, 10 µA low).

CACHE and CPUDIS are the only unusual signals on the cache connector. CACHE, an active-high signal, disables the memory controller (MDU) so that it cannot start a memory cycle and allows the cache card to supply the data instead. The active-high transition of the CACHE signal must occur at the same time as the active-low transition of the /AS signal, or earlier. Asserting CACHE prevents the memory controller from beginning a RAM, ROM, or NuBus cycle. If CACHE is asserted after the memory controller has started a cycle, that cycle is not affected. Also, CACHE does not affect memory controller cycles for I/O devices.

Since CACHE must be asserted at /AS, the cache controller leaves CACHE unasserted except when the cache is not active (that is, /CIOUT is asserted and /CENABLE is deasserted, or an alternate bus master owns the bus as indicated by an asserted /BGACK signal).

The CPUDIS signal is used during diagnostic testing to disable the MC68030 on the main logic board and tristate its outputs. An emulator in the cache card can assert CPUDIS and, after waiting for the end of the current bus cycle, drive all signals.

◆ *Note:* NuBus cards can access each other without that transaction appearing on the CPU bus. This can lead to inconsistency between memory on the NuBus card, for example, and the cached version of that memory. For this reason, the Macintosh Operating System always marks the NuBus address space noncacheable, as controlled by the MC68030 processor's on-chip PMMU.

The /BGACK signal is not driven high quickly enough by the main logic board to satisfy cache memory operations. Your card design should include a 2.2 kΩ resistor to pull /BGACK up to +5 V, and a circuit to double-rank synchronize /BGACK before using it. You can double-rank synchronize /BGACK by putting it through two DQ flip-flops that are clocked by the CPUCLK signal, and using the output from the second flip-flop.

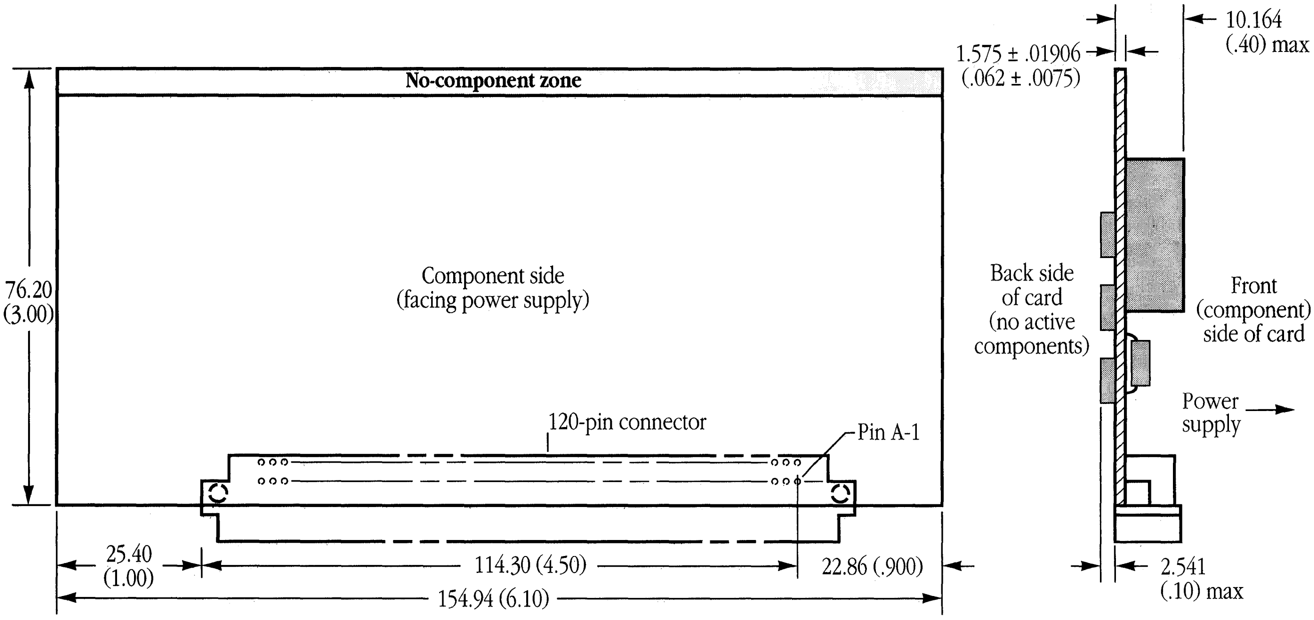
△ **Important** Although the cache expansion connector is capable of other functions, Apple plans to support its use for RAM cache cards only. △

Mechanical design guidelines for the cache card

Figure 23-2 shows two views of the cache card. The larger drawing is a component-side view showing the maximum dimensions and the location of the 120-pin connector. Note that the location of the connector is given with reference to the edge of the connector, not to pin A1. The size limitation is required for proper cooling of the card. If you fail to adhere to these guidelines, your design could create a potential reliability problem for the customer.

To the right is an end view (from the front of the computer) showing the card thickness and component placement. Notice that the maximum card thickness is 1.575 ±0.1906 mm (0.062 ±0.0075 inch).

- **Figure 23-2** Cache card design guide



Dimensions are in millimeters with inches in parentheses.

Here are some guidelines you should follow when designing your cache card hardware to ensure proper thermal dissipation and eliminate the possibility of electrical interference with the ROM SIMM.

- Card warpage must be controlled to within a 2.541 mm (0.10 inch) deviation from the ideal.
- You should place no components or traces in the top 3.811 mm (0.150 inch) of the card, on either the front (component) side or the back side.
- Component height must not extend beyond the edge of the card in any direction.
- Component height cannot exceed 10.164 mm (0.40 inch) on the front side of the board (toward the power supply).
- No component or wire lead on the back side of the card (toward the ROM SIMM) may extend more than 2.541 mm (0.10 inch) from the surface of the card.
- You should place all active components on the front (component) side of the card; you may place resistors and capacitors that do not exceed the height limitation on the back side of the card.

Power consumption guidelines

Table 23-5 gives current allocation for a cache card in the Macintosh IIci and compares it with the current allotted to each NuBus card. Exceeding these guidelines could create a potential reliability problem in the host system.

■ **Table 23-5** Comparison of current limits for a Macintosh IIci cache card and a NuBus card

Nominal power supply value, V	**Cache card maximum current, A**	**NuBus card maximum current, A**
+5	1.0	2.0
+12	Not available	0.175
–12	Not available	0.150

Most of the information in Appendix A on EMI, heat dissipation, and product safety also applies to a cache memory expansion card. One noticeable difference is that the maximum power dissipation for a Macintosh IIci cache card is 5 W versus the 7.5 W specified for an expansion card in a conventional PDS.

Appendix A EMI, Heat Dissipation, and Product Safety Guidelines

This appendix provides general information that you should become familiar with before you start your expansion card design. Guidelines are given for electromagnetic interference (EMI), heat dissipation, and product safety standards. These guidelines apply to both NuBus and PDS expansion cards.

EMI guidelines for expansion cards

Every Macintosh-family computer meets FCC radio and television electromagnetic interference (EMI) requirements as a stand-alone device, or when connected to a peripheral device such as a printer or modem. However, you should follow certain guidelines when designing your expansion card to avoid exceeding the mandatory FCC limits when the card is installed in the computer. This section provides EMI design guidelines for the following configurations:

- when an expansion card is mounted internally with no external I/O connections
- when an expansion card is mounted internally *and* external I/O connections are provided

These guidelines apply equally to expansion cards designed for NuBus slots and processor-direct slots.

Without external I/O connections

The following guidelines should enable you to build an add-on that does not degrade the computer to the extent that the combination product will not meet FCC regulations for Class B equipment. However, you are responsible for the FCC authorization of the combination product. Development testing should be undertaken as soon as you have completed a realistic expansion card in order to alert you, the developer, to any serious EMI problems. You can resolve these problems by rerouting signal conductors, filtering and bypassing, and terminating buses properly to eliminate excessive transient ringing (undershoot) on clocks and other signals. You must use appropriate emission control techniques on the card and on wiring to any connectors for external I/O.

- Use cards with four layers (power and ground on two of the layers), or with extremely low impedance busing of power and ground lines, to reduce EMI pickup and emanations.
- Buffer high-speed signals and separate them from lower-speed circuitry.
- Buffer signals from the expansion connector as close to the connector as possible and limit the drive to one LS load with a maximum capacitance of 18 pF.
- Make internal interconnecting cables as short as possible. Position cables such that inductive and capacitive coupling with the computer's subassemblies is minimized. You should not bundle conductors carrying high-speed signals with conductors carrying low-speed signals. In certain cases, you may have to use internal shielding or twisted pairs within cables.

- Do not locate high-speed components such as clock oscillators and their signal lines near the expansion port connector and shield.
- Provide good high-frequency decoupling in addition to adequate power supply filtering at the low-voltage power connectors of the card. These precautions avoid degrading the low emission levels conducted from the computer's 120 VAC power connector.

With external I/O connections

Macintosh computers with NuBus have external I/O connections for each NuBus slot. Macintosh computers that use a processor-direct slot for expansion have only one external I/O connection, called the *external device access opening.* In general, these guidelines apply equally to both configurations. When there are differences they are noted.

Connecting a cable to an external I/O connector can seriously compromise the emissions integrity of the computer. You are likely to exceed allowable limits on conducted or radiated emissions unless you take care during construction and test *the total system as it will be operated.* The total system includes

- the unmodified computer
- the expansion card and all internal cables used to modify the computer (and, for a Macintosh SE, Macintosh SE/30, and Macintosh IIsi, a connector card)
- the external cable and peripherals to be connected

It is very important for you to do the following:

- Follow all the guidelines given for internal expansion cards as described previously.
- Include EMI filtering in each I/O line and power line going to or beyond (outside) the I/O connector. This is best achieved by using deglitch packs (RC or LC networks) or common mode chokes located directly at the connector.
- Shape the spectrum of signals, especially video, in the frequency domain so that unrequired bandwidths and harmonics are not needlessly propagated.
  (Note: Computer designers tend to prefer very fast edges so that timing errors are never a problem, but it is these very fast edges that cause high-amplitude harmonics in the frequency domain and lead to emission problems.)

- Use a connector of high quality, one that has high-conductivity (electrical) plating and accepts a shielded plug. The tin-plated steel DB series of connectors is one obvious example. The connector on a NuBus card should be mechanically and electrically bonded to the metal I/O fence on the rear of the expansion card. The connector on a PDS card should be mechanically and electrically bonded to the metal chassis behind the external device access opening at the rear of the computer. An unsecured, unbonded connector protruding through the opening is almost sure to cause a major EMI problem.
- External metal conductor cables must be shielded, without exception. Solder bond the entire circumference of the braided shield to provide a low-impedance path to the entire perimeter of the connector.
- Interconnecting cables should be as short as possible. Do not bundle conductors carrying high-speed signals with conductors carrying low-speed signals. In certain cases, you may have to use internal shielding or twisted pairs within cables.

Heat dissipation guidelines

Macintosh expansion cards, by their own heat dissipation, produce increased temperatures within the computer. Because excessive heat can have a detrimental effect on performance and reliability, Apple has developed a general set of heat dissipation guidelines for each of the two categories of expansion cards, NuBus and processor-direct slot.

Heat dissipation guidelines for NuBus cards

You should follow these guidelines when designing NuBus cards for a Macintosh computer with the NuBus expansion interface:

- Dissipation by a NuBus expansion card of up to 13.3 W of power provides a comfortable margin for the major components. This total is arrived at as follows:

+5 V	@	2.0 A	=	10.0 W
+12 V	@	0.175 A	=	2.1 W
–12 V	@	0.1 A	=	1.2 W
Total power			=	13.3 W

Dissipation of more than 13.3 W by a card may cause excessive temperature rise on certain critical components. Apple studies indicate that at an ambient temperature of about 24°C, 13.3 W of dissipated power from the expansion card will cause an acceptable rise in average component case temperature to about 53°C. (Studies were conducted with an internal hard disk drive installed.)

- You can achieve optimum cooling for both the logic board and expansion cards by keeping the expansion card as short as possible; the minimum possible is 7.0 inches (see the section "Card Description" in Chapter 6). In addition, placing larger components near the bottom side of the expansion card is desirable.
- Place hot components on the expansion card directly against the card; they should have the widest possible printed wiring traces. This provides for better cooling as the airflow from the fan moves from the rear of the computer to the front.
- Installation of an expansion card should not cause the case temperature of an internal hard disk to rise more than 20°C over external ambient air temperature.

Heat dissipation guidelines for PDS cards

You should follow these guidelines when designing an expansion card for a Macintosh computer with the PDS expansion interface:

- Dissipation by the expansion card of up to 7.5 W provides a comfortable margin for the major computer components. Dissipation of more than 7.5 W may cause excessive temperature rise on certain critical components. Apple studies indicate that at an ambient temperature of about 24°C, 7.5 W of dissipated power from the expansion card will cause an acceptable rise in average component case temperature to about 53°C for the main logic board components located directly under the expansion card (studies were conducted with an internal hard disk drive installed). Note that the cache card in the Macintosh IIci is an exception: maximum heat dissipation for this card is only 5 W.
- The most heat-sensitive logic board components include the microprocessor and the DRAM SIMM modules. The maximum recommended temperature for the center of the microprocessor case is 65°C. The maximum recommended temperature for the case of each component on the DRAM SIMM modules is 60°C.
- You can achieve optimum cooling for both the logic board and expansion card in a Macintosh SE, for example, by positioning the expansion card as far above the logic board as possible (while still avoiding mechanical interference with the chassis); the suggested distance is 16.8 mm. In addition, you will get a more uniform temperature distribution in the Macintosh SE if you place the components on the top (away from the main logic board) rather than the bottom side of the card.

- Put hot components toward the rear of the expansion card, away from the front bezel, to get better cooling by the airflow from the fan.
- An expansion card should not cause the case temperature of an internal hard disk to rise more than 15°C over external ambient air temperature.

Product safety

Every Macintosh computer meets national and international product safety requirements. Therefore, any additional cards and components need careful safety consideration to maintain the same degree of electrical and mechanical safety. When you design an expansion card to fit inside a Macintosh computer, you must consider several product safety issues.

American (Underwriters Laboratories—UL), Canadian (Canadian Standards Association—CSA), and European (Institute for Industrial Research and Standards—IIRS) regulatory organizations give product safety approval to a Macintosh computer with a NuBus interface with dummy expansion cards in each NuBus slot. The same agencies give approval to a Macintosh computer with a PDS interface without any expansion card (including a dummy card) installed in the slot. When you change the design of either product by adding a functional expansion card, and resell the unit, the product becomes delisted. Technically, you should resubmit the computer with your card (or cards) installed and have the new (combination) product evaluated. The new product should have a new model number, and the computer essentially becomes a component of your system.

You can maintain product safety if you follow these guidelines:

- Stay within the maximum power specification of the expansion connector.
- Use components that have been certified by the safety agencies. Components such as lithium batteries, power relays, tape drives, disk drives, fans, wires and cables, and other parts should have at least UL and CSA approvals.
- Properly secure (mount) the components. Avoid mountings that depend on adhesive only or mountings that allow movement of components or cards.
- Avoid using materials that could contribute to a fire. This includes PCB material, card guides, and other parts. In general, PCB material should be flame rated 94V-1 or better; wire should be UL Listed/CSA Certified, flame rated VW-1; and plastic parts within the enclosure should be flame rated 94V-2 or better.
- Place PCBs and other components so that they do not block vent openings or fan circulation.

- Secure all wiring and provide chafing protection to prevent degradation of the insulation on moving parts or sharp edges.
- Do not configure connectors such that a hazard is created if they are plugged in backward or into the wrong connector.
- Verify that the installation or conversion kits are complete. Provide any special tools required for installation or conversion (for example, a special nut driver). Provide any special hardware; don't expect the installer to modify (bend or drill, for example) existing hardware.
- Make sure installation or conversion instructions are complete. Provide a review by a person who is unfamiliar with the product to ensure that instructions are complete and accurate enough for that person to understand.
- Avoid splicing wires. Your conversion kit should provide new harnesses if they are required.
- Avoid soldering. If soldering is necessary, the connection should be made mechanically secure before soldering (no tack soldering).

The following guidelines apply particularly to expansion cards that use *high voltages.*

- Do not allow maintenance work to be performed by someone unfamiliar with the hazards involved. If the Macintosh has a built-in CRT, repair personnel must be aware of the dangers of shock from the primary, the charge stored on the CRT, and the implosion potential of the CRT.
- Be careful to maintain proper through-air and over-surface spacings between the high-voltage components (power supply, relays, a built-in CRT, and so forth) and the logic circuitry. Remember that spacings are measured under worst-case conditions and that if a card can be moved, spacings will be measured with the card in the worst position. Spacing tables can be found in the following safety standards: UL478, CSA 22.2 No. 154-M1983, CSA 22.2 No. 220-M1986, IEC 380, IEC 435, and IEC 950.
- Maintain proper insulation thickness or layers between the high-voltage components and the logic circuitry. (Proper insulation is defined in the standards listed in the preceding item.) If a low-voltage circuit can contact a high-voltage wire, the low-voltage wire must also be insulated for the higher voltage.
- Don't place components next to high-voltage parts.

Appendix B **Sample Video Card Firmware**

This appendix contains a sample of the declaration ROM firmware code for Macintosh video cards using Macintosh Programmer's Workshop assembly language. Notice that this sample code reflects the configuration of the declaration ROM's firmware shown in Chapter 8. Also included in this appendix are samples of the primary initialization code and the secondary initialization code for a video card.

```
;-------------------------------------------------------------------------------
;
; (c) Apple Computer, Inc.        1986-1991
; All rights reserved.
;
;-------------------------------------------------------------------------------
;
;      File: SampleRom.a
;
;      This is a sample configuration ROM source for a NuBus and/or PDS Macintosh
;      Video Card. It demonstrates most of the features and capabilities of video
;      cards, including multiple monitor support, system feature identification, and
;      manufacturer-specific data structures.
;
;      This fictitious card supports the following modes:
;
;            1) 640x480 at 1,2,4,8bpp          ( 8-bit QuickDraw)
;            2) 640x480 at 1,2,4,8,16,32bpp    (32-bit QuickDraw)
;            3) 640x480 at 1bpp fixed          (Family mode for 2)
;
;            4) 640x870 at 1,2,4,8bpp          (8/32-bit QuickDraw)
;
;-------------------------------------------------------------------------------

;Include files

       PRINT  OFF

       INCLUDE      'SysEqu.a'          ; MPW Assembler Equates
       INCLUDE      'SysErr.a'          ;
       INCLUDE      'Traps.a'           ;
       INCLUDE      'VideoEqu.a'        ;
       INCLUDE      'ROMEqu.a'          ;
       INCLUDE      'QuickEqu.a'        ;
       INCLUDE      'SlotEqu.a'         ;

       INCLUDE      'DepVideoEqu.a'     ; Left as an exercise for the reader

       PRINT  ON

       MACHINE      MC68020

DclROM       MAIN

;===============================================================================
; Constants
;===============================================================================
;
;
; sResource constants are in this form :  sRsrc_Vidxyz.
;
;      Where  x = size:       S - 640x480, B - 640x870.
;             y = addressing: 8-bit QD or 32-bit QD.
;             z = option:     F - family mode or none.
;
```

```
; These are the constants for the sRsrcs that are in the sRsrc directory.
; The numbering is developed in this fashion to facilitate PrimaryInit:
;
;       Bit 7 = 1 always on for video sRsrc IDs (i.e., functional sRsrcs are ≥ 128).
;       Bit 6 = 1 if optional 1-bit only mode, 0 otherwise.
;       Bit 5 = 0
;       Bit 4 = 0
;       Bit 3 = 1 if 640x870 screen, 0 for 640x480.
;       Bit 2 = 0
;       Bit 1 = 0
;       Bit 0 = 1 if accessed in 32-bit mode (32-bit QD), else 0 for 24-bit mode
        (8-bit QD).

sRsrc_Board         EQU      1               ; board sResource (>0 & <128)

sRsrc_VidS8         EQU    $80               ; functional sResources
sRsrc_VidS32        EQU    $81               ;
sRsrc_VidB8         EQU    $88               ;
sRsrc_VidS32F       EQU    $C1               ;

;===================================================================================
;       Directory (must be in ascending order!)
;===================================================================================

_sRsrcDir
              OSLstEntry   sRsrc_Board,_sRsrc_Board          ; board sRsrc List
              OSLstEntry   sRsrc_VidS8,_sRsrc_VidS8          ; video sRsrc List
              OSLstEntry   sRsrc_VidS32,_sRsrc_VidS32        ; video sRsrc List
              OSLstEntry   sRsrc_VidB8,_sRsrc_VidB8          ; video sRsrc List
              OSLstEntry   sRsrc_VidS32F,_sRsrc_VidS32F      ; video sRsrc List
              DatLstEntry  EndOfList,0                       ; end of list

;===================================================================================
;       sRsrc_Board list
;===================================================================================

       STRING       C

_sRsrc_Board
              OSLstEntry   sRsrcType,_BoardType        ; offset to board descriptor
              OSLstEntry   sRsrcName,_BoardName        ; offset to name of board
              OSLstEntry   sRsrcName,_VidICON          ; offset to icon
              DatLstEntry  BoardId,SampleBoardID       ; board ID # (assigned by DTS)
              OSLstEntry   PrimaryInit,_sPInitRec      ; offset to PrimaryInit exec blk
              OSLstEntry   VendorInfo,_VendorInfo      ; offset to vendor info record
              OSLstEntry   SecondaryInit,_sSInitRec    ; offset to SecondaryInit block
              OSLstEntry   sRsrcVidNames,_sVidNameDir ; video name directory
              DatLstEntry  EndOfList,0                 ; end of list

;
; Each NuBus/PDS board has a single board identifier, even if it contains
; multiple devices.
;

_BoardType
              DC.W         CatBoard                   ; board sResource
              DC.W         TypBoard                   ;
              DC.W         0                          ;
              DC.W         0                          ;

_BoardName
              DC.L         'Sample Video Card'        ; name of board

;
```

```
; You can optionally include an ICON, cicn, icl8, or icl4 sResource that Monitors
; will display instead of its default ICON.  We just have an ICON resource here
; it is the ICON for Classic Macintoshes.
;

_VidICON
             DC.L   $3FFFE000,$40001000,$4FFF9000,$50005000       ; ICON sResource
             DC.L   $55505000,$50005000,$55005000,$50007FFC
             DC.L   $54004002,$50004FF2,$5000500A,$5000554A
             DC.L   $4FFFD00A,$4000550A,$3FFFD00A,$0800540A
             DC.L   $3BFF500A,$2000540A,$3FFFD00A,$0000500A
             DC.L   $3FFE500A,$40014FF2,$4FF94002,$50053FFC
             DC.L   $55050810,$50057BDE,$54054002,$50057FFE
             DC.L   $50050000,$4FF90000,$40010000,$3FFE0000

;
; The video name directory associates an optional name string with each video sRsrc
; present. This name is read by Monitors and is presented in a video mode family
; selector in the Options dialog box.  If your card does not have any mode families,
; it does not need a name directory, unless you would just like Monitors to identify
; your monitor type(s) by name.
;

_sVidNameDir
             OSLstEntry   sRsrc_VidS32,_sNameReg      ; name rcd for regular mode
             OSLstEntry   sRsrc_VidS32F,_sNameOption  ; name rcd for special mode
             DatLstEntry  EndOfList,0

_sNameReg
             DC.L         EndNameReg-_sNameReg
             DC.W         NameRegResID                ; localization resID
             DC.B         'Full Color Display'        ;
             ALIGN  2
EndNameReg

_sNameOption
             DC.L         EndNameOption-_sNameOption
             DC.W         NameOptResID                ; localization resID
             DC.B         'Black & White Only'        ;
             ALIGN  2
EndNameOption

;====================================================================================
;      PrimaryInit record
;====================================================================================

_sPInitRec
             DC.L         _EndsPInitRec-_sPInitRec  ; physical block size
             INCLUDE      'SamplePrimaryInit.a'     ; the header/code
             ALIGN  2
_EndsPInitRec
```

```
;====================================================================================
;      SecondaryInit record
;====================================================================================

_sSInitRec
             DC.L         _EndsSInitRec-_sSInitRec  ; physical block size
             INCLUDE      'SampleSecondaryInit.a'   ; the header/code
             ALIGN  2
_EndsSInitRec

;====================================================================================
;      Vendor info record
;====================================================================================
;
; The vendor information record allows the developer to include revision level and
part ; number information in their ROMs.
;

             STRING       C

_VendorInfo
             OSLstEntry   VendorId,_VendorId        ; offset to vendor ID
             OSLstEntry   RevLevel,_RevLevel        ; offset to revision
             OSLstEntry   PartNum,_PartNum          ; offset to part number record
             OSLstEntry   Date,_Date                ; offset to ROM build date
             DatLstEntry  EndOfList,0               ;
_VendorId
             DC.L         'Apple Computer, Inc.'    ; vendor ID
_RevLevel
             DC.L         'Sample 2.0'              ; revision level
_PartNump
             DC.L         'DAF/jmp'                 ; part number
_Date
             DC.B         '&SysDate'                ; date


;====================================================================================
;      sRsrc_Video
;====================================================================================

_sRsrc_VidS8
             OSLstEntry   sRsrcType,_VideoType      ; video type descriptor
             OSLstEntry   sRsrcName,_VideoName      ; offset to driver name string
             OSLstEntry   sRsrc_DrvrDir,_VidDrvrDir ; offset to driver directory
             DatLstEntry  sRsrc_HWDevId,1           ; hardware device ID

             OSLstEntry   MinorBaseOS,_MinorBase    ; offset to frame buffer array
             OSLstEntry   MinorLength,_MinorLength  ; offset to frame buffer length

             OSLstEntry   sGammaDir,_GammaDirS      ; directory for 640x480 monitor

;Parameters
             OSLstEntry   FirstVidMode,_OBMs        ; offset to OneBitMode parms
             OSLstEntry   SecondVidMode,_TBMs       ; offset to TwoBitMode parms
             OSLstEntry   ThirdVidMode,_FBMs        ; offset to FourBitMode parms
             OSLstEntry   FourthVidMode,_EBMs       ; offset to EightBitMode parms
             DatLstEntry  EndOfList,0               ; end of list

;------------------------------------------------------------------------------------

_sRsrc_VidS32
             OSLstEntry   sRsrcType,_VideoType      ; Mac OS video type descriptor
             OSLstEntry   sRsrcName,_VideoName      ; offset to driver Name string
             OSLstEntry   sRsrc_DrvrDir,_VidDrvrDir ; offset to driver directory
             DatLstEntry  sRsrcFlags,6              ; f32BitMode & fOpenAtStart set
             DatLstEntry   sRsrc_HWDevId,1           ; hardware device ID

             OSLstEntry    MinorBaseOS,_MinorBase    ; offset to frame buffer array
             OSLstEntry    MinorLength,_MinorLength  ; offset to frame buffer length

             OSLstEntry    sGammaDir,_GammaDirS      ; directory for 640x480 monitor

;Parameters
             OSLstEntry    FirstVidMode,_OBMs        ; offset to OneBitMode parms
             OSLstEntry    SecondVidMode,_TBMs       ; offset to TwoBitMode parms
             OSLstEntry    ThirdVidMode,_FBMs        ; offset to FourBitMode parms
             OSLstEntry    FourthVidMode,_EBMs       ; offset to EightBitMode parms
             OSLstEntry    FifthVidMode,_D16BMs      ; offset to SixteenBitMode parms
             OSLstEntry    SixthVidMode,_D32BMs      ; offset to ThirtyTwoBitMode
parms
             DatLstEntry   EndOfList,0               ; end of list

;-------------------------------------------------------------------------------------

_sRsrc_VidB8
             OSLstEntry    sRsrcType,_VideoType      ; Mac OS video type descriptor
             OSLstEntry    sRsrcName,_VideoName      ; offset to driver name string
             OSLstEntry    sRsrc_DrvrDir,_VidDrvrDir ; offset to driver directory
             DatLstEntry   sRsrc_HWDevId,1           ; hardware device ID

             OSLstEntry    MinorBaseOS,_MinorBase    ; offset to frame buffer array
             OSLstEntry    MinorLength,_MinorLength  ; offset to frame buffer length

             OSLstEntry    sGammaDir,_GammaDirB      ; directory for 640x870 monitor

;Parameters
             OSLstEntry    FirstVidMode,_OBMb        ; offset to OneBitMode parms
             OSLstEntry    SecondVidMode,_TBMb       ; offset to TwoBitMode parms
             OSLstEntry    ThirdVidMode,_FBMb        ; offset to FourBitMode params
             OSLstEntry    FourthVidMode,_EBMb       ; offset to EightBitMode params
             DatLstEntry   EndOfList,0               ; end of list

;-------------------------------------------------------------------------------------
;
; This video sResource demonstrates video family modes. As an alternate to the full
; function resource above, it doesn't make too much sense, but this alternate could
; have had different screen size or display characteristics. Although 32-bit
addressing
; is not required for a one-bit only display, the new Slot Manager is required to
allow
; alternate video sResources.  Also, note that we are pointing to a different set of
; parameters for this mode.  Since this is a black & white device only, we are
; pretending that it is a fixed-CLUT device.
;
_sRsrc_VidS32F
             OSLstEntry    sRsrcType,_VideoType      ; Mac OS video type descriptor
             OSLstEntry    sRsrcName,_VideoName      ; offset to driver name string
             OSLstEntry    sRsrc_DrvrDir,_VidDrvrDir ; offset to driver directory
             DatLstEntry   sRsrcFlags,6              ; f32BitMode & fOpenAtStart set
             DatLstEntry   sRsrc_HWDevId,1           ; hardware device ID

             OSLstEntry    MinorBaseOS,_MinorBase    ; offset to frame buffer array
             OSLstEntry    MinorLength,_MinorLength  ; offset to frame buffer length

             OSLstEntry    sGammaDir,_GammaDirS      ; directory for 640x480 monitor

;Parameters
             OSLstEntry    FirstVidMode,_OBMFs       ; offset to OneBitMode fixed parms
             DatLstEntry   EndOfList,0               ; end of list

;-------------------------------------------------------------------------------------
```

```
             STRING       C

_VideoType
             DC.W         CatDisplay                ; <Category>
             DC.W         TypVideo                  ; <Type>
             DC.W         DrSwApple                 ; <DrvrSw>
             DC.W         DrHwSample                ; <DrvrHw>

_VideoName
             DC.L         'Display_Video_Apple_Samp'; video drive name

_MinorBase
             DC.L         defMinorBase              ; frame buffer offset

_MinorLength
             DC.L         defMinorLength            ; frame buffer length

;====================================================================================
;      Driver directories
;====================================================================================

_VidDrvrDir
             OSLstEntry   sMacOS68020,_sMacOS68020  ; driver directory for Mac OS
             DatLstEntry  EndOfList,0               ;

_sMacOS68020
             DC.L         _End020Drvr-_sMacOS68020  ; physical block size
             INCLUDE      'SampleDrvr.a'            ; driver code
_End020Drvr

;====================================================================================
;      Gamma directories
;====================================================================================

             STRING       C

_GammaDirS                                          ; for the 640x480 monitor
             OSLstEntry   128,_SmallGamma
             DatLstEntry  EndOfList,0

_GammaDirB                                          ; for the 640x870 monitor
             OSLstEntry   128,_BigGamma
             DatLstEntry  EndOfList,0

_SmallGamma
             DC.L         _EndStdGamma-_StdGamma

             DC.W         SGammaResID
             DC.B         'Small Gamma'             ; Monitors name
             ALIGN        2
             DC.W         $0000                     ; gVersion
             DC.W         drHwSample                ; gType
             DC.W         $0000                     ; gFormulaSize
             DC.W         $0001                     ; gChanCnt
             DC.W         $0100                     ; gDataCnt
             DC.W         $0008                     ; gChanWidth

             DC.L         $0005090B,$0E101315,$17191B1D,$1E202224
             DC.L         $2527282A,$2C2D2F30,$31333436,$37383A3B
             DC.L         $3C3E3F40,$42434445,$4748494A,$4B4D4E4F
             DC.L         $50515254,$55565758,$595A5B5C,$5E5F6061
             DC.L         $62636465,$66676869,$6A6B6C6D,$6E6F7071
             DC.L         $72737475,$76777879,$7A7B7C7D,$7E7F8081
             DC.L         $81828384,$85868788,$898A8B8C,$8C8D8E8F
             DC.L         $90919293,$94959596,$9798999A,$9B9B9C9D
             DC.L         $9E9FA0A1,$A1A2A3A4,$A5A6A6A7,$A8A9AAAB
             DC.L         $ABACADAE,$AFB0B0B1,$B2B3B4B4,$B5B6B7B8
             DC.L         $B8B9BABB,$BCBCBDBE,$BFC0C0C1,$C2C3C3C4
             DC.L         $C5C6C7C7,$C8C9CACA,$CBCCCDCD,$CECFD0D0
             DC.L         $D1D2D3D3,$D4D5D6D6,$D7D8D9D9,$DADBDCDC
             DC.L         $DDDEDFDF,$E0E1E1E2,$E3E4E4E5,$E6E7E7E8
             DC.L         $E9E9EAEB,$ECECEDEE,$EEEFF0F1,$F1F2F3F3
             DC.L         $F4F5F5F6,$F7F8F8F9,$FAFAFBFC,$FCFDFEFF

_EndSmallGamma

_BigGamma
             DC.L         _EndBigGamma-_BigGamma
             DC.W         BGammaResID
             DC.B         'Big Gamma'                ; Monitors name
             ALIGN        2
             DC.W         $0000                      ; gVersion
             DC.W         drHwSample                 ; gType
             DC.W         $0000                      ; gFormulaSize
             DC.W         $0003                      ; gChanCnt
             DC.W         $0100                      ; gDataCnt
             DC.W         $0008                      ; gChanWidth

             DC.L         $0005090B,$0E101315,$17191B1D,$1E202224      ; red channel
             DC.L         $2527282A,$2C2D2F30,$31333436,$37383A3B
             DC.L         $3C3E3F40,$42434445,$4748494A,$4B4D4E4F
             DC.L         $50515254,$55565758,$595A5B5C,$5E5F6061
             DC.L         $62636465,$66676869,$6A6B6C6D,$6E6F7071
             DC.L         $72737475,$76777879,$7A7B7C7D,$7E7F8081
             DC.L         $81828384,$85868788,$898A8B8C,$8C8D8E8F
             DC.L         $90919293,$94959596,$9798999A,$9B9B9C9D
             DC.L         $9E9FA0A1,$A1A2A3A4,$A5A6A6A7,$A8A9AAAB
             DC.L         $ABACADAE,$AFB0B0B1,$B2B3B4B4,$B5B6B7B8
             DC.L         $B8B9BABB,$BCBCBDBE,$BFC0C0C1,$C2C3C3C4
             DC.L         $C5C6C7C7,$C8C9CACA,$CBCCCDCD,$CECFD0D0
             DC.L         $D1D2D3D3,$D4D5D6D6,$D7D8D9D9,$DADBDCDC
             DC.L         $DDDEDFDF,$E0E1E1E2,$E3E4E4E5,$E6E7E7E8
             DC.L         $E9E9EAEB,$ECECEDEE,$EEEFF0F1,$F1F2F3F3
             DC.L         $F4F5F5F6,$F7F8F8F9,$FAFAFBFC,$FCFDFEFF

             DC.L         $0005090B,$0E101315,$17191B1D,$1E202224      ; green channel
             DC.L         $2527282A,$2C2D2F30,$31333436,$37383A3B
             DC.L         $3C3E3F40,$42434445,$4748494A,$4B4D4E4F
             DC.L         $50515254,$55565758,$595A5B5C,$5E5F6061
             DC.L         $62636465,$66676869,$6A6B6C6D,$6E6F7071
             DC.L         $72737475,$76777879,$7A7B7C7D,$7E7F8081
             DC.L         $81828384,$85868788,$898A8B8C,$8C8D8E8F
             DC.L         $90919293,$94959596,$9798999A,$9B9B9C9D
             DC.L         $9E9FA0A1,$A1A2A3A4,$A5A6A6A7,$A8A9AAAB
             DC.L         $ABACADAE,$AFB0B0B1,$B2B3B4B4,$B5B6B7B8
             DC.L         $B8B9BABB,$BCBCBDBE,$BFC0C0C1,$C2C3C3C4
             DC.L         $C5C6C7C7,$C8C9CACA,$CBCCCDCD,$CECFD0D0
             DC.L         $D1D2D3D3,$D4D5D6D6,$D7D8D9D9,$DADBDCDC
             DC.L         $DDDEDFDF,$E0E1E1E2,$E3E4E4E5,$E6E7E7E8
             DC.L         $E9E9EAEB,$ECECEDEE,$EEEFF0F1,$F1F2F3F3
             DC.L         $F4F5F5F6,$F7F8F8F9,$FAFAFBFC,$FCFDFEFF
```

```
                DC.L        $0005090B,$0E101315,$17191B1D,$1E202224      ; blue channel
                DC.L        $2527282A,$2C2D2F30,$31333436,$37383A3B
                DC.L        $3C3E3F40,$42434445,$4748494A,$4B4D4E4F
                DC.L        $50515254,$55565758,$595A5B5C,$5E5F6061
                DC.L        $62636465,$66676869,$6A6B6C6D,$6E6F7071
                DC.L        $72737475,$76777879,$7A7B7C7D,$7E7F8081
                DC.L        $81828384,$85868788,$898A8B8C,$8C8D8E8F
                DC.L        $90919293,$94959596,$9798999A,$9B9B9C9D
                DC.L        $9E9FA0A1,$A1A2A3A4,$A5A6A6A7,$A8A9AAAB
                DC.L        $ABACADAE,$AFB0B0B1,$B2B3B4B4,$B5B6B7B8
                DC.L        $B8B9BABB,$BCBCBDBE,$BFC0C0C1,$C2C3C3C4
                DC.L        $C5C6C7C7,$C8C9CACA,$CBCCCDCD,$CECFD0D0
                DC.L        $D1D2D3D3,$D4D5D6D6,$D7D8D9D9,$DADBDCDC
                DC.L        $DDDEDFDF,$E0E1E1E2,$E3E4E4E5,$E6E7E7E8
                DC.L        $E9E9EAEB,$ECECEDEE,$EEEFF0F1,$F1F2F3F3
                DC.L        $F4F5F5F6,$F7F8F8F9,$FAFAFBFC,$FCFDFEFF

_EndBigGamma
```

```
;=====================================================================================
;      One-bit per pixel parameters
;=====================================================================================
;
_OBMs
             OSLstEntry   mVidParams,_OBVParms       ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages1s           ; number of video pages
             DatLstEntry  mDevType,CLUTType          ; device type
             DatLstEntry  EndOfList,0                ; end of list

_OBMFs
             OSLstEntry   mVidParams,_OBVParms       ; offset to vid parameters
             OSLstEntry   mTable,_OBVFixedCLUT       ; offset to fixed table
             DatLstEntry  mPageCnt,Pages1s           ; number of video pages
             DatLstEntry  mDevType,FixedType         ; device type
             DatLstEntry  EndOfList,0                ; end of list

_OBMb
             OSLstEntry   mVidParams,_OBVParmb       ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages1b           ; number of video pages
             DatLstEntry  mDevType,CLUTType          ; device type
             DatLstEntry  EndOfList,0                ; end of list

_OBVParms
             DC.L         _EndOBVParms-_OBVParms     ; physical block size

             DC.L         defmBaseOffset             ; QuickDraw base offset
             DC.W         RB1s                       ; physRowBytes
             DC.W         defmBounds_Ts,defmBounds_Ls,defmBounds_Bs,defmBounds_Rs
             DC.W         defVersion                 ; bmVersion
             DC.W         0                          ; packType not used
             DC.L         0                          ; packSize not used
             DC.L         defmHRes                   ; bmHRes
             DC.L         defmVRes                   ; bmVRes
             DC.W         ChunkyIndexed              ; bmPixelType
             DC.W         1                          ; bmPixelSize
             DC.W         1                          ; bmCmpCount
             DC.W         1                          ; bmCmpSize
             DC.L         defmPlaneBytes             ; bmPlaneBytes
_EndOBVParms

_OBVFixedCLUT
             DC.L         _EndOBVFixedCLUT-_OBVFixedCLUT   ; physical block size

             DC.L         $0000                      ; ctSeed
             DC.W         $0000                      ; ctFlags
             DC.W         $0001                      ; ctSize (n - 1)

             DC.W         $0000, $FFFF,$FFFF,$FFFF   ; value, r,g,b (white entry)
             DC.W         $0001, $0000,$0000,$0000   ; value, r,g,b (black entry)
_EndOBVFixedCLUT

_OBVParmb
             DC.L         _EndOBVParmb-_OBVParmb     ; physical block size

             DC.L         defmBaseOffset             ; QuickDraw base offset
             DC.W         RB1b                       ; physRowBytes
             DC.W         defmBounds_Tb,defmBounds_Lb,defmBounds_Bb,defmBounds_Rb
             DC.W         defVersion                 ; bmVersion
             DC.W         0                          ; packType not used
             DC.L         0                          ; packSize not used
             DC.L         defmHRes                   ; bmHRes
             DC.L         defmVRes                   ; bmVRes
             DC.W         ChunkyIndexed              ; bmPixelType
             DC.W         1                          ; bmPixelSize
             DC.W         1                          ; bmCmpCount
             DC.W         1                         ; bmCmpSize
             DC.L         defmPlaneBytes            ; bmPlaneBytes
_EndOBVParmb

;==================================================================================
;      Two-bit per pixel parameters
;==================================================================================
;
_TBMs
             OSLstEntry   mVidParams,_TBVParms      ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages2s          ; number of video pages
             DatLstEntry  mDevType,CLUTType         ; device type
             DatLstEntry  EndOfList,0               ; end of list

_TBMb
             OSLstEntry   mVidParams,_TBVParmb      ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages2b          ; number of video pages
             DatLstEntry  mDevType,CLUTType         ; device type
             DatLstEntry  EndOfList,0               ; end of list

_TBVParms
             DC.L         _EndTBVParms-_TBVParms    ; physical block size

             DC.L         defmBaseOffset            ; QuickDraw base offset
             DC.W         RB2s                      ; physRowBytes
             DC.W         defmBounds_Ts,defmBounds_Ls,defmBounds_Bs,defmBounds_Rs
             DC.W         defVersion                ; bmVersion
             DC.W         0                         ; packType not used
             DC.L         0                         ; packSize not used
             DC.L         defmHRes                  ; bmHRes
             DC.L         defmVRes                  ; bmVRes
             DC.W         ChunkyIndexed             ; bmPixelType
             DC.W         2                         ; bmPixelSize
             DC.W         1                         ; bmCmpCount
             DC.W         2                         ; bmCmpSize
             DC.L         defmPlaneBytes            ; bmPlaneBytes
_EndTBVParms

_TBVParmb
             DC.L         _EndTBVParmb-_TBVParmb    ; physical block size

             DC.L         defmBaseOffset            ; QuickDraw base offset
             DC.W         RB2b                      ; physRowBytes
             DC.W         defmBounds_Tb,defmBounds_Lb,defmBounds_Bb,defmBounds_Rb
             DC.W         defVersion                ; bmVersion
             DC.W         0                         ; packType not used
             DC.L         0                         ; packSize not used
             DC.L         defmHRes                  ; bmHRes
             DC.L         defmVRes                  ; bmVRes
             DC.W         ChunkyIndexed             ; bmPixelType
             DC.W         2                         ; bmPixelSize
             DC.W         1                         ; bmCmpCount
             DC.W         2                         ; bmCmpSize
             DC.L         defmPlaneBytes            ; bmPlaneBytes
_EndTBVParmb

;==================================================================================
;      Four-bit per pixel parameters
;==================================================================================

_FBMs
             OSLstEntry   mVidParams,_FBVParms      ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages4s          ; number of video pages
             DatLstEntry  mDevType,CLUTType         ; device type
             DatLstEntry  EndOfList,0               ; end of list
```

```
_FBMb
             OSLstEntry   mVidParams,_FBVParmb       ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages4b           ; number of video pages
             DatLstEntry  mDevType,CLUTType          ; device type
             DatLstEntry  EndOfList,0                ; end of list

_FBVParms
             DC.L         _EndFBVParms-_FBVParms     ; physical block size

             DC.L         defmBaseOffset             ; QuickDraw base offset
             DC.W         RB4s                       ; physRowBytes
             DC.W         defmBounds_Ts,defmBounds_Ls,defmBounds_Bs,defmBounds_Rs
             DC.W         defVersion                 ; bmVersion
             DC.W         0                          ; packType not used
             DC.L         0                          ; packSize not used
             DC.L         defmHRes                   ; bmHRes
             DC.L         defmVRes                   ; bmVRes
             DC.W         ChunkyIndexed              ; bmPixelType
             DC.W         4                          ; bmPixelSize
             DC.W         1                          ; bmCmpCount
             DC.W         4                          ; bmCmpSize
             DC.L         defmPlaneBytes             ; bmPlaneBytes
_EndFBVParms

_FBVParmb
             DC.L         _EndFBVParmb-_FBVParmb     ; physical block size

             DC.L         defmBaseOffset             ; QuickDraw base offset
             DC.W         RB4b                       ; physRowBytes
             DC.W         defmBounds_Tb,defmBounds_Lb,defmBounds_Bb,defmBounds_Rb
             DC.W         defVersion                 ; bmVersion
             DC.W         0                          ; packType not used
             DC.L         0                          ; packSize not used
             DC.L         defmHRes                   ; bmHRes
             DC.L         defmVRes                   ; bmVRes
             DC.W         ChunkyIndexed              ; bmPixelType
             DC.W         4                          ; bmPixelSize
             DC.W         1                          ; bmCmpCount
             DC.W         4                          ; bmCmpSize
             DC.L         defmPlaneBytes             ; bmPlaneBytes
_EndFBVParmb
```

```
;===================================================================================
;     Eight-bit per pixel parameters
;===================================================================================

_EBMs
             OSLstEntry   mVidParams,_EBVParms      ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages8s          ; number of video pages
             DatLstEntry  mDevType,CLUTType         ; device type
             DatLstEntry  EndOfList,0               ; end of list

_EBMb
             OSLstEntry   mVidParams,_EBVParmb      ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages8b          ; number of video pages
             DatLstEntry  mDevType,CLUTType         ; device type
             DatLstEntry  EndOfList,0               ; end of list
EBVParms
             DC.L         _EndEBVParms-_EBVParms    ; physical block size

             DC.L         defmBaseOffset            ; QuickDraw base offset
             DC.W         RB8s                      ; physRowBytes
             DC.W         defmBounds_Ts,defmBounds_Ls,defmBounds_Bs,defmBounds_Rs
             DC.W         defVersion                ; bmVersion
             DC.W         0                         ; packType not used
             DC.L         0                         ; packSize not used
             DC.L         defmHRes                  ; bmHRes
             DC.L         defmVRes                  ; bmVRes
             DC.W         ChunkyIndexed             ; bmPixelType
             DC.W         8                         ; bmPixelSize
             DC.W         1                         ; bmCmpCount
             DC.W         8                         ; bmCmpSize
             DC.L         defmPlaneBytes            ; bmPlaneBytes
_EndEBVParms

_EBVParmb
             DC.L         _EndEBVParmb-_EBVParmb    ; physical block size

             DC.L         defmBaseOffset            ; QuickDraw base offset
             DC.W         RB8b                      ; physRowBytes
             DC.W         defmBounds_Tb,defmBounds_Lb,defmBounds_Bb,defmBounds_Rb
             DC.W         defVersion                ; bmVersion
             DC.W         0                         ; packType not used
             DC.L         0                         ; packSize not used
             DC.L         defmHRes                  ; bmHRes
             DC.L         defmVRes                  ; bmVRes
             DC.W         ChunkyIndexed             ; bmPixelType
             DC.W         8                         ; bmPixelSize
             DC.W         1                         ; bmCmpCount
             DC.W         8                         ; bmCmpSize
             DC.L         defmPlaneBytes            ; bmPlaneBytes
_EndEBVParmb
```

```
;=====================================================================================
;      Direct mode (16-bit per pixel) parameters
;=====================================================================================

_D16BMs
             OSLstEntry   mVidParams,_DB16VParms     ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages16s          ; number of video pages
             DatLstEntry  mDevType,DirectType        ; direct device type
             DatLstEntry  EndOfList,0                ; end of list

_DB16VParms
             DC.L         _EndDBVParms-_DB16VParms   ; physical block size
             DC.L         defmBaseOffset             ; QuickDraw base offset
             DC.W         RB16s                      ; physRowBytes
             DC.W         defmBounds_Ts,defmBounds_Ls,defmBounds_Bs,defmBounds_Rs
             DC.W         defVersion                 ; bmVersion
             DC.W         0                          ; packType not used
             DC.L         0                          ; packSize not used
             DC.L         defmHRes                   ; bmHRes
             DC.L         defmVRes                   ; bmVRes
             DC.W         ChunkyDirect               ; bmPixelType
             DC.W         16                         ; bmPixelSize
             DC.W         3                          ; bmCmpCount
             DC.W         5                          ; bmCmpSize
             DC.L         defmPlaneBytes             ; bmPlaneBytes
_EndDB16VParms

;=====================================================================================
;      Direct mode (32-bit per pixel) parameters
;=====================================================================================

_D32BMs
             OSLstEntry   mVidParams,_DBVParms       ; offset to vid parameters
             DatLstEntry  mPageCnt,Pages32s          ; number of video pages
             DatLstEntry  mDevType,DirectType        ; direct device type
             DatLstEntry  EndOfList,0                ; end of list

_DB32VParms
             DC.L         _EndDB32VParms-_DB32VParms ; physical block size
             DC.L         defmBaseOffset
             DC.W         RB32s                      ; physRowBytes
             DC.W         defmBounds_Ts,defmBounds_Ls,defmBounds_Bs,defmBounds_Rs
             DC.W         defVersion                 ; bmVersion
             DC.W         0                          ; packType not used
             DC.L         0                          ; packSize not used
             DC.L         defmHRes                   ; bmHRes
             DC.L         defmVRes                   ; bmVRes
             DC.W         ChunkyDirect               ; bmPixelType
             DC.W         32                         ; bmPixelSize
             DC.W         3                          ; bmCmpCount
             DC.W         8                          ; bmCmpSize
             DC.L         defmPlaneBytes             ; bmPlaneBytes
_EndDB32VParms
```

```
;==================================================================================
;     Format/header block
;==================================================================================

             WITH         FHeaderRec
             ORG          ROMSize-FHeaderRec.fhBlockSize

             DC.L         (_sRsrcDir-*)**$00FFFFFF  ;offset to sResource directory
             DC.L         ROMSize                   ;length of declaration data
             DC.L         0                         ;CRC {Patched by crcPatch}
             DC.B         2                         ;revision (1-9)
             DC.B         AppleFormat               ;format
             DC.L         TestPattern               ;test pattern
             DC.B         0                         ;reserved byte
             DC.B         $E1                       ;ByteLanes: 1111 0001

             ENDWITH
             END
```

```
;---------------------------------------------------------------------------
;
; (c) Apple Computer, Inc.      1986-1991
; All rights reserved.
;
;---------------------------------------------------------------------------
;
;      File: SamplePrimaryInit.a
;
;      This is the primary initialization code for the Sample Video Card
;      source.  PrimaryInit for video cards serves a number of functions:
;
;            1) initializes the video frame buffer and video output
;            2) disables VBL interrupts
;            3) displays a 50% dithered gray pattern on the screen
;            4) performs any maintainence on Slot Manager structures
;
;      This sample card supports a number of configurations.  These include
;      640x480 (for both 8-bit & 32-bit QuickDraw), 640x870, and, on machines
;      that have 32-bit QD in ROM, a 640x480 mode that supports only one-bit
;      per pixel to demonstrate video mode families.  This fictitious card
;      can detect the two different types of displays or the lack of any
;      connected monitor via hardware sense lines.
;
;      Of particular interest are the sections where the code determines
;      that the configuration has changed, the way it selects new defaults,
;      and the relationship with SecondaryInit when running 32-bit addressed
;      sResources.
;
;      Sections that are hardware-specific are not included in this listing;
;      their place in code is marked with the tag <DEVICE-SPECIFIC>.
;
;---------------------------------------------------------------------------
-
;      Header
;---------------------------------------------------------------------------
-

            DC.B        sExec_2                 ; code revision
            DC.B        sCPU_68020              ; CPU type is 68020
            DC.W        0                       ; reserved
            DC.L        Begin1stInit-*          ; offset to code

            WITH        seBlock,spBlock

Begin1stInit

;
; Set initial vendor status
;

            MOVE.W      #seSuccess,seStatus(A0)   ; assume a good return
```

```
;
; Form 32-bit base address into A1.  (Make sure to swap into 32-bit addressing
; mode before using this address.)
;

             MOVE.L            #$F0000000,D1        ; D1 <- F0000000
             MOVE.B            seSlot(A0),D0        ; get slot number
             BFINS       D0,D1{4:4}                 ; D1 <- Fs000000
             MOVE.L            D1,A1                ; copy to address reg

;
; <DEVICE-SPECIFIC>
;
; Disable VBL interrupts here.  They will be reactivated at _Open time.
;

             BSR         DisableVBLs                ; left as an exercise for the
reader
;
;
; <DEVICE-SPECIFIC>
;
; Read the connected display type.  This may be from sense lines that identify
; the connected monitor, DIP switches on the card, or setup information in the
; slot PRAM (note that the System file is not open yet).  Often video
; generated at the wrong timing is not viewable, so dynamically reading the
; connected monitor is always preferred.  Also, it is desirable to identify
; the absence of a connected monitor so that it will not show up on the
; desktop.
;
; In this example, we assume that the display type will be returned in D1.  If
; the display is 640x480, D1=$80; if 640x870, D1=$88.  These are the 24-bit
; (8-bit QD) flavored spIDs for small and large display sRsrc lists. If no display
; is connected, this routine returns D1=$FF, which is not an applicable spID. Later,
; we will set the spID to disabled in D7, so set it to $FF for now.
;

             BSR         ReadSenseLines             ; left as an exercise for the
reader
;
; <DEVICE-SPECIFIC>
;
; Initialize the video generation and CLUT here. You may init to any video mode,
; but it is preferable to initialize to 1bpp mode if the card has that
; capability, or to the default screen depth as called out in your video
; sResource.  Also, gray the screen with a 50% dithered gray pattern.  Note that
; this routine should probably do nothing when D1=$FF (i.e., no-connect).
;

             BSR         InitScreen                 ; left as an exercise for the
reader

;
; Set up a slot parameter block for sRsrc pruning.  At startup, all sResources
; in the sResource directory are loaded.  After PrimaryInit, only one active
; video sResource must be present, so all others must be removed.
;

             SUBA        #spBlockSize,SP            ; make an spBlock
             MOVE.L            SP,A0                ; get pointer to parms
             MOVE.B            D0,spSlot(A0)        ; identify the slot
             CLR.B       spExtDev(A0)               ; external device = 0

;
; Read the slot pRAM to determine what the currently saved mode is.  The first
; word is the board ID, followed by the default pixdepth.  This code keeps the
; spID of the video sResource in VendorUse2.  This is an important part of
; the implementation of video mode families.  Later, in PrimaryInit, this mode
; is tested for compatibility with the current display, and, if it is, it is
; made the enabled mode.
;
; Since we can always have a 24-bit addressed (8-bit QD) video sRsrc, we simply match
; the monitor type if 32-bit QuickDraw isn't around.  If this frame buffer can only
; work in 32-bit addressing, then we return a magic value ($8001) in seResult, which
; disables this card until SecondaryInit if the new Slot Manager is loaded as a
; system patch.
;
; The boot screen (the one the happy Mac is on) is set to the depth in VendorUse1
; before the System file is open.  The other screens are set up per the 'scrn'
; resource by the system.
;

             SUBA         #SizesPRAMRec,SP            ; block for PRAM record
             MOVE.L              SP,spResult(A0)      ; point to it
             _sReadPRAMRec                            ; read it
             MOVE.B              VendorUse2(SP),D4    ; get default spID

             BTST         #3,D4                       ; was this a big screen?
             BNE.S        @isBig                      ; yes
             CMP.B        #$80,D1                     ; compare to actual display
             BEQ.S        SetUp                       ; monitor hasn't changed
             BRA.S        Changed                     ;
@isBig
             CMP.B        #$88,D1                     ; compare to actual display
             BEQ.S        SetUp                       ; monitor hasn't changed

;
; If we got here, then the monitor isn't the same type that we had last time.
; We need to resetup PRAM and perform other invalidations.
;

Changed
             MOVE.B              D1,D4                ; copy the default to D4
             MOVE.B              D1,VendorUse2(SP)    ; make it the default
             MOVE.B              #FirstVidMode,VendorUse1(SP) ; select default depth

             MOVE.L              spResult(A0),spsPointer(A0)   ; set up parameter
block
             _SPutPRAMRec                             ; write the new record out

             ADDA         #SizesPRAMRec,SP            ; eliminate PRAM block

;
; Let's go for it. We've invalidated as necessary.  The 24-bit spID for the
; desired configuration is in D4.  We will test for the presence of 32-bit
; QuickDraw and the new Slot Manager. If 32-bit QD is not in ROM (remember,
; the System file is not yet open), then we will convert to using the 24-bit
; addressed spID and fix it in SecondaryInit. If the new Slot Manager is
; present in ROM, then we will also add the 1-bit only sRsrc as an alternate,
; disabled mode.
;
```

```
SetUp

; Here is the 32-bit QuickDraw ID sequence

             MOVE.L              #$A89F,D0           ; _Unimplemented trap
             _GetTrapAddress ,NewTool                ;
             MOVE.L              A0,D1               ; save result
             MOVE.L              #$AB03,D0           ; now test for new QD
             _GetTrapAddress ,NewTool                ; get it too
             CMPA.L              D1,A0               ; are they the same?
             BNE.S        Make32                     ; if ≠, then QD32
             BCLR         #0,D4                      ; if no QD32, then clear bit
             BRA.S        SlotTst                    ;

; If the 32-bit stuff is present, then pick the enhanced mode

Make32
             CMP.B        #sRsrc_VidS32F,D4          ; is it the special mode?
             BEQ.S        @1                         ; yes
             MOVE.B              #sRsrc_VidS32,D4    ; set 32-bit mode as active
             MOVE.B              #sRsrc_VidS32F,D7   ; make one-bit the alternate
             BRA.S        SlotTst                    ;
@1
             MOVE.B       #sRsrc_VidS32,D7           ; set full mode the alternate

;
; Here is the Slot Manager version ID. If the original Slot Manager is present,
; then this routine returns an error code in D0. If we don't have the new Slot
; Manager, then set D7 back to $FF, since we can't disable.
;

SlotTst
             _SVersion                               ; what Slot Manager do we have?
             BEQ.S        Prune                      ; new, so continue

             MOVE.B              #$FF,D7             ; make disable invalid

;
; Time to massage the sRsrc directory.  The spIDs for all modes are concatenated into
; a long word of possible modes. Each byte is compared to D4, and if not the valid
; mode, its sRsrc is deleted. In addition, each nybble is tested to see if it
; should be disabled.
;

Prune
             MOVE.L              #$808188C1,D6       ; the mode list (four modes)
             MOVEQ        #3,D1                      ; counter in D1 (zero based)
@0           MOVE.B              D6,D0               ; get lowest byte
             LSR.L        #8,D6                      ; rotate list
             CMP.B        D7,D0                      ; should this mode be disabled?
             BNE.S        2                          ; if not, then continue
             MOVE.B              D0,spID(A0)         ; mark this ID for disabling
             MOVE.L              #1,spParamData(A0)  ; set to one to disable
             _SetsRsrcState                          ; disable it
             CLR.L        spParamData(A0)            ; no longer needed
             BRA.S        @10                        ; and continue
@2           CMP.B        D4,D0                      ; is this the valid mode?
             BEQ.S        @10                        ; yup, so skip deletion
             MOVE.B              D0,spID(A0)         ; set the mode
@5           _sDeleteSRTRec                          ; remove the invalid entry
@10          DBRA         D1,@0

;
; clean up spBlock on stack
;
```

```
CleanUp
             ADDA          #spBlockSize,SP            ; flush the block

;
; return to your regularly scheduled start code
;

             RTS

             ENDWITH
```

```
;-----------------------------------------------------------------------------
;
; (c) Apple Computer, Inc.       1986-1991
; All rights reserved.
;
;-----------------------------------------------------------------------------
;
;      File  : SampleSecondaryInit.a
;
;      This is the secondary initialization code for the Sample Video Card
;      source.  SecondaryInits are one-time initialization code that run
;      immediately after the system patches have been loaded.  Video cards
;      may want to perform certain reconfigurations now that the new
;      Slot Manager and 32-bit QuickDraw have had an opportunity to load.
;
;      At this point in the boot process, the boot screen has been opened
;      and has a gDevice, but the other displays have only had PrimaryInit.
;      For many cards, this code only performs housekeeping tasks with
;      declaration structures.  Hardware setup is generally complete at
;      PrimaryInit.
;
;      Video mode families depend upon inactivated video sResource lists
;      to be present.  Since this is a new Slot Manager feature,
;      older machines will depend upon the addition of inactive modes
;      at SecondaryInit after the Slot Manager patch is loaded.
;
;      Remember that interrupts are enabled for the boot device, but not
;      for other devices that have not yet been opened.
;

;-----------------------------------------------------------------------------
;      Header
;-----------------------------------------------------------------------------

             DC.B         sExec_2                   ; code revision
             DC.B         sCPU_68020                ; CPU type is 68020
             DC.W         0                         ; reserved
             DC.L         Begin2ndInit-*            ; offset to Secondary Init code

             WITH         seBlock,spBlock

Begin2ndInit
;
; Always a successful return
;
             MOVE.W       #seSuccess,seStatus(A0)   ; VendorStatus <- 1


;
; Calculate the 32-bit base address now while we have the slot number.
; (Make sure to swap into 32-bit addressing mode before using this address.)
;

             MOVE.L              #$F0000000,D1      ; D1 <- F0000000
             MOVE.B              seSlot(A0),D0      ; get slot number
             BFINS        D0,D1{4:4}                ; D1 <- Fs000000
             MOVE.L              D1,A1              ; copy to address reg


;
; Set up a slot parameter block
;

             SUBA         #spBlockSize,SP           ; make a slot parameter block
             MOVE.L              SP,A0              ; get pointer to parm block now
             MOVE.B              D0,spSlot(A0)      ; put slot in pBlock
             CLR.B        spExtDev(A0)              ; no external devices
```

```
;
; If the new Slot Manager was present in ROM, then activations and deactivation have
; already been performed. We also reverify that 32-bit QuickDraw is around.
;

             _sVersion                                ; find the Slot Manager version
             CMP.L       #2,spResult(A0)              ; get the result (1=RAM, 2=ROM)
             BEQ.S       SecInitDone                  ; if in ROM, done

             MOVE.L            #$A89F,D0              ; the unimplemented trap
             _GetTrapAddress ,NewTool                 ; get its address
             MOVE.L             A0,D1                 ; save it
             MOVE.L            #$AB03,D0              ; the 32-bit QD trap
             _GetTrapAddress ,NewTool                 ; get it address
             CMPA.L             D1,A0                 ; if not implemented
             BEQ.S       SecInitDone                  ; then no QD32

;
; Find the currently active video sResource. This is the one that was active upon
; completion of PrimaryInit. It may be either 24- or 32-bit addressed.
;

             MOVE.L             SP,A0                 ; point at spBlock again
             CLR.B       pID(A0)                      ; start search at id=0
             CLR.B       spTBMask(A0)                 ; we're going for an exact match
             MOVE.W      #catDisplay,spCategory(A0) ; look for: Display,
             MOVE.W      #typVideo,spCType(A0)        ;     Video,
             MOVE.W      #drSwApple,spDrvrSW(A0)      ;     Apple,
             MOVE.W      #drHwSample,spDrvrHW(A0)     ;     Sample.
             _sNextTypesRsrc                          ; get the spsPointer
             BNE.S       SecInitDone                  ; (If failed, we’re in trouble.)
             MOVE.W      spRefNum(A0),D5              ; save driver refNum for later

;
; If the big screen sResource is installed, then we are done since it doesn't have
; a 32-bit addressed counterpart.
;

             CMP.B       #sRsrc_VidB8,spID(A0)        ; is it the big screen?
             BEQ.S       SecInitDone                  ; yes, so done

;
; Read the slot pRAM rec to find out the current desired mode. The VendorUse2 byte of
; slot PRAM contains the spID of the target video sRsrc list, which will be either
; the full-function sResource or the 1-bit only one. We will delete the 24-bit
; sRsrc list and add the other lists, enabled or disabled appropriately. We know
; that it will not already be set to a 32-bit sRsrc since we would have done that in
; PrimaryInit if the new Slot Manager were present.
;

             SUBA        #SizesPRAMRec,SP             ; allocate block for PRAM record
             MOVE.L      SP,spResult(A0)              ; point to it
             _sReadPRAMRec                            ; read it
             MOVE.B      VendorUse2(SP),D4            ; get the current spID
             ADDA.W      #SizesPRAMRec,SP             ; free PRAM record buffer

;
; Delete the 24-bit version (its spID is still in the spBlock)
;

             _sDeleteSRTRec

;
; Insert the desired mode (its spID is in D4) as an active sResource
;
```

```
             MOVE.B              D4,spID(A0)         ; add this sRsrc in
             CLR.L        spParamData(A0)            ; clear to enable activation
             CLR.L        spsPointer(A0)             ; add back an sRsrc in directory
             _sInsertSRTRec                          ; add it back in

;
; We can determine the spID of the alternate mode by flipping bit 6. We know we need
; to add this mode in deactivated.
;

             BCHG         #6,D4                      ; get the alternate mode
             MOVE.L              #1,spParamData(A0)  ; set up to disable it
             MOVE.B              D4,spID(A0)         ; put in block
             _sInsertSRTRec                          ; insert and disable

;
; Test if we are the boot screen. If we are, we need to update the gDevice. The
; dCtlDevBase is fixed by the Slot Manager.
;

             SUBQ         #4,SP                      ; make room for function return
             _GetDeviceList                          ; get the boot gDevice
             MOVE.L              (SP)+,A0            ; get the gdHandle
             MOVE.L              (A0),A0             ; get pointer to gDevice
             CMP.W        gdRefNum(A0),D5            ; was this the boot device
             BNE.S        SecInitDone                ; no, so quit

             MOVE.L              gdPMap(A0),A0       ; get pixMap handle
             MOVE.L              (A0),A0             ; get pixMap ptr
             MOVE.L              A1,pmBaseAddr(A0)   ; save new base address

;
; Return to your regularly scheduled start code
;
SecInitDone
             ADDA         #spBlockSize,SP            ; flush the block
             RTS
             ENDWITH
```

Appendix C **Video Card Driver Example**

This appendix provides an example of a possible video card driver, written in Macintosh Programmer's Workshop assembly language.

```
;------------------------------------------------------------------------------------
;
; (c) Apple Computer, Inc. 1986-1991
; All rights reserved.
;
;------------------------------------------------------------------------------------
;
;     File  : SampleDrvr.a
;
;     This file contains a sample video driver for use by the Macintosh
;     OS in MPW 3.x format.  It is structured as a normal slot device driver.
;     It is included as part of the declaration ROM image, so this file does
;     not have its own INCLUDE statements.
;
;     Hardware-specific sections of the driver are not included here; their place
;     in code is marked with the tag  <DEVICE-SPECIFIC>.
;
;     This fictitious video card supports 1-, 2-, 4-, 8-, 16-, and 32-bit/pixel on
;     640x480 and 1-, 2-, 4-, and 8-bpp on 640x870 displays.  Also, in systems that
;     have the new version of the Slot Manager and 32-bit QuickDraw, there is a 1-bit
;     only sRsrc for the 640x480 display, implemented as a video mode family. The system
;     configuration and capabilities were determined at PrimaryInit and SecondaryInit,
;     so the driver performs only minimal identification.
;

             BLANKS             ON
             STRING             ASIS
             MACHINE      MC68020

;------------------------------------------------------------------------------------
;     Local data storage declarations and flag word equates
;------------------------------------------------------------------------------------

; This is device storage that is stored in the dCtlStorage field of the DCE

VidLocals    RECORD             0
saveMode     DS.W         1                          ; the current mode setting
savePage     DS.W         1                          ; the current video page setting
saveBaseAddr DS.L         1                          ; the current screen base
address
saveSQElPtr  DS.L         1                          ; ptr to slot interrupt queue
elem
saveGammaPtr DS.L         1                          ; pointer to the current gamma
table
saveVidParms DS.L         1                          ; pointer to video config data
GFlags             DS.W         1                    ; flags word
VidLocalSize EQU          *-VidLocals                ; size of this record structure
             ENDR

; Flags within GFlags word

GrayFlag     EQU          15                         ; luminance mapped if set
IntDisFlag   EQU          14                         ; interrupts disabled if set
DirectFlag   EQU          13                         ; direct type pixel mode if set

;------------------------------------------------------------------------------------
;     Video driver header
;------------------------------------------------------------------------------------

VidDrvr      DC.W         $4C00                      ; ctl, status, needsLock
             DC.W         0,0,0                      ; not an ornament

; Entry point offset table

             DC.W         VideoOpen-VidDrvr          ; open routine
             DC.W         VidDrvr-VidDrvr           ; no prime
             DC.W         VideoCtl-VidDrvr          ; control
             DC.W         VideoStatus-VidDrvr       ; status
             DC.W         VideoClose-VidDrvr        ; close

;
; It is important to include the driver version number here. The card driver is
; opened using the VideoName string in the declaration ROM structures. The _Open
; call looks in the current resource chain, and if it finds an appropriately named
; driver (resource type 'DRVR', resID does not matter) with a higher version number,
; then it substitutes that driver's code for the driver included in the video ROM.
; Third-party developers cannot generally utilize this mechanism for overrides,
; since their driver would have to be in the System file, but should support this
; driver version number mechanism in their own system extensions. Note also that
; the driver's name is a Pascal string in the driver, but is a C string with the
; leading period omitted in the configuration data.
;

             STRING       Pascal
VideoTitle   DC.B         '.Display_Video_Apple_Samp ; video driver name
             STRING             ASIS
             ALIGN        2                         ; make sure word aligned
             DC.W         DrvrROMVersion            ; driver version number

;-------------------------------------------------------------------------------------
;
; VideoOpen allocates and initializes private storage for the device. It identifies
the ; configuration set up at Primary/SecondaryInit.  It installs the default gamma
table. ; Finally, it installs the interrupt handler and enables the interrupts.
;
; Entry:     A0 = csParam block pointer
;            A1 = DCE pointer
;
;-------------------------------------------------------------------------------------

             WITH         VidLocals,SlotIntQElement,spBlock

VideoOpen
;
; Allocate private storage (since block is CLEAR, GFlags are zeroed) and get a
; pointer to it in A3
;

             MOVEQ        #VidLocalSize,D0          ; get size of parameters
             _ResrvMem     ,SYS                     ; make room as low as possible
             MOVEQ        #VidLocalSize,D0          ; get size of parameters
             _NewHandle    ,SYS,CLEAR               ; get some memory for VidLocals
             BNE                OpError             ; => return an error in open
             MOVE.L             A0,dCtlStorage(A1)  ; save returned handle in DCE
             _HLock                                 ; and lock it down
             MOVE.L              (A0),A3            ; get a pointer to it
```

```
;
; Find the current video spID by using the Slot Manager to search for this device,
; which was set up at boot by PrimaryInit. This will identify the exact hardware and
; software configuration that we are running with at this time.
;

             SUBA         #spBlockSize,SP            ; get a slot parameter block
                                                     ; pointer
             MOVE.L              SP,A0               ; get pointer to block in A0
             MOVE.B              dCtlSlot(A1),spSlot(A0)    ; copy the slot number
from the
                                                     ; DCE
             CLR.B        spID(A0)                   ; start looking at spID=0
             CLR.B        spExtDev(A0)               ; no external devices
             CLR.B        spHWDev(A0)                ; only one hardware device here
             CLR.B        spTBMask(A0)               ; we're going for an exact
match
             MOVE.W       #catDisplay,spCategory(A0) ; look for:Display,
             MOVE.W       #typVideo,spCType(A0)       ;     Video,
             MOVE.W       #drSwApple,spDrvrSW(A0)     ;     Apple,
             MOVE.W       #drHwSample,spDrvrHW(A0)    ;     Sample.
             _sNextTypesRsrc                          ; get the spsPointer
             BNE          OpError1                    ; if ≠, then there has been a
                                                      ;   serious error
;
; Point to the appropriate set of video parameters for this mode.  We contrived the
; spIDs so that if 32-bit QD is available bit 0 is set, if the 640x870 display is
; connected then bit 3 is set, and if we are in the special one-bit only 640x480
; mode, then bit 6 is set.
;

             MOVE.B              spID(A0),D0         ; get the spID
             BTST         #6,D0                      ; test the special bit
             BNE.S        @RealSmall                 ; if set, then special mode
             BTST         #3,D0                      ; test the big screen bit
             BNE.S        @ThinkBig                  ; if ≠, then we have a big
screen
             BTST         #0,D0                      ; is it the 32-bit or 24-bit
                                                     ; flavor?
             BNE.S        @Medium                    ;
             LEA          Small24Parms,A2            ; general parameters for 640x480
                                                     ;   displays
             BRA.S        @cont1                     ;
@RealSmall
             LEA          OneParms,A2                ; general parameters for 1-bit
                                                     ;   only mode
             BRA.S        @cont1                     ;
@Medium
             LEA          Small32Parms,A2            ; general parameters for direct
                                                     ;   mode display
             BRA.S        @cont1                     ;
@ThinkBig    LEA          BigParms,A2                ; general parameters for 640x870
                                                     ;   displays
@cont1
             MOVE.L              A2,saveVidParms(A3  ; save these in private storage

;
; Load the default gamma table from the slot resource list.  Each video sRsrc list
; includes a directory of gamma tables that are appropriate for this mode.  We will
; set the default gamma table from this directory, which always has an spID of 128.
; A0 still contains the current video sResource information. If no gamma directory
; is present, the software should just make an uncorrected, linear gamma table.
;
```

```
             MOVE.B              #sGammaDir,spID(A0); look for the gamma directory
             _sFindStruct                           ; get gamma directory's
                                                    ; spsPointer
             MOVE.B              #128,spID(A0)      ; get default gamma table
             _sGetBlock                             ; we want a ptr in sysheap

; skip over gamma table header

             MOVE.L              spResult(A0),A0    ; point to head of the block
             ADDA         #2,A0                     ; skip resID
@Name        TST.B        (A0)+                     ; skip over gamma name
             BNE.S        @Name                     ;
             ADDA         #1,A0                     ; word align pointer
             MOVE.L              A0,D0              ; get in d-reg
             AND.L        #$FFFFFFFE,D0             ; round it
             MOVE.L              D0,saveGammaPtr(A3 ; put it in private storage
             ADDA         #spBlockSize,SP           ; release the Slot Manager block
;
; Get and install the interrupt handler. Call the SetInterrupt utility code to do
; this.  This utility also starts the interrupts going. If there is an error
; condition, EnableVGuts returns with Z-bit set.
;
             MOVEQ        #sqHDSize,D0              ; allocate a slot queue element
             _NewPtr      ,SYS,CLEAR                ; get it from system heap
cleared
             BNE.S        OpError1                  ;
             MOVE.L              A0,saveSQElPtr(A3) ; save the queue element

             BSR          EnableVGuts               ; do it
             BNE.S        OpError2                  ;
;
; All done!
;
             MOVEQ        #noErr,D0                 ; no error
EndOpen      RTS                                    ; return

OpError2     MOVE.L              saveSQElPtr(A3),A0 ; get slot interrupt queue
                                                    ; element
             _DisposPtr                             ; release it
OpError1     MOVE.L              dCtlStorage(A1),A0 ; dispose the private storage
             _DisposHandle                          ; release it
OpError      MOVE.L              #OpenErr,D0        ; say can't open driver
             BRA.S        EndOpen

             ENDWITH
```

```
;------------------------------------------------------------------------------------
;
; Video Driver Control Call Handler
;
;      (0)  Reset
;      (1)  KillIO
;      (2)  SetMode
;      (3)  SetEntries
;      (4)  SetGamma
;      (5)  GrayPage
;      (6)  SetGray
;      (7)  SetInterrupt
;      (8)  DirectSetEntries
;      (9)  SetDefaultMode
;
;   Entry:   A0     = IO parameter block pointer
;            A1     = DCE pointer
;   Uses:    A2     = cs parameters (ie. A2 <- csParam(A0))
;            A3     = pointer to private storage
;            A4     = scratch (must be preserved)
;            D0-D3  = scratch (don't need to be preserved)
;
;   Exit:    D0    = error code
;
;-----------------------------------------------------------------------------

VideoCtl     MOVE.L              A0,-(SP)                  ; save work registers
                                                     ;   (A0 is saved
                                                     ;   because it is used by
                                                     ;   ExitDrvr)
             MOVE.W              csCode(A0),D0       ; get the opCode
             MOVE.L              csParam(A0),A2      ; A2 <- ptr to control
parameters
             MOVE.L              dCtlStorage(A1),A3  ; get pointer to private storage
             MOVE.L              (A3),A3

             CMP.W        #9,D0                      ; IF csCode NOT IN [0..9] THEN
             BHI.S        CtlBad                     ;   error, csCode out of bounds
             MOVE.W              CtlJumpTbl(PC,D0.W*2),D0  ; get the routine’s
relative
                                                     ; offset
             JMP          CtlJumpTbl(PC,D0.W)        ; jump to it

CtlJumpTbl   DC.W         VidReset-CtlJumpTbl        ; $00 => VidReset
             DC.W         VidKillIO-CtlJumpTbl       ; $01 => VidKillIO
             DC.W         SetVidMode-CtlJumpTbl      ; $02 => SetVidMode
             DC.W         SetEntries-CtlJumpTbl      ; $03 => SetEntries
             DC.W         SetGamma-CtlJumpTbl        ; $04 => SetGamma
             DC.W         GrayPage-CtlJumpTbl        ; $05 => GrayPage
             DC.W         SetGray-CtlJumpTbl         ; $06 => SetGray
             DC.W         SetInterrupt-CtlJumpTbl    ; $07 => SetInterrupt
             DC.W         CtlBad-CtlJumpTbl          ; $08 => DirectSetEntries
             DC.W         SetDefaultMode-CtlJumpTbl ; $09 => SetDefault mode

CtlBad              MOVEQ        #controlErr,D0      ; else say we don't do this one
             BRA.S        CtlDone                    ; and return

CtlGood      MOVEQ        #noErr,D0                  ; return no error

CtlDone      MOVE.L              (SP)+,A0            ; restore registers
             BRA          ExitDrvr
```

```
VidReset
;-----------------------------------------------------------------------------------
;
; Reset the card to its default.  For the sample card, reset video to the
; default depth, page zero, and gray the screen.
;
;-----------------------------------------------------------------------------------

             WITH         VidLocals,VDPageInfo

             MOVE.W              #FirstVidMode,D1    ; get the default mode
identifier
             MOVE.W              D1,csMode(A2)       ; return default mode
             MOVE.W              D1,saveMode(A3)     ; remember it

             MOVEQ        #0,D0                      ; get the default page value
             MOVE.W              D0,savePage(A3)     ; save page zero as current page
             MOVE.W              D0,csPage(A2)       ; return the page

             BSR          HWSetDepth                 ; set the depth from D1
             BSR          HWSetPage                  ; set the page from D0

             MOVE.L              D0,saveBaseAddr(A3); save the new base address
             MOVE.L              D0,csBaseAddr(A2)   ; return the base address
             MOVE.W              csPage(A2),D0       ; setup D0 for GrayScreen
             BSR          GrayScreen                 ; paint the screen gray

             BRA.S        CtlGood                    ; => no error

             ENDWITH

VidKillIO
;-----------------------------------------------------------------------------------
;
; This routine is not normally required by video cards, but if you support
; "asynchronous" writes to the CLUT as part of your slot interrupt handler, then
; this call should be implemented to immediately flush the pending changes to the
; CLUT hardware and clear any state flags.
;
;-----------------------------------------------------------------------------------

             BRA.S        CtlGood

SetVidMode
;-----------------------------------------------------------------------------------
;
;   Set the card to the specified mode and page.
;   If either is invalid, returns badMode error.
;
;   If the card is already set to the specified mode, then do nothing.
;
;-----------------------------------------------------------------------------

             WITH         VidLocals,VDPageInfo

             MOVE.W              csMode(A2),D1       ; D1 = mode
             BSR          ChkMode                    ; check mode
             BNE.S        CtlBad                     ; => not a valid mode

             MOVE.W              csPage(A2),D0       ; D0 = page
             BSR          ChkPage                    ; check page
             BNE.S        CtlBad                     ; => not a valid page

; Only set the mode if it has changed
```

```
SetDepth
             MOVE.W             csMode(A2),D2       ; D2 = mode
             CMP          saveMode(A3),D2           ; has the mode changed?
             BEQ.S        ModeOK1                   ; => no, check the page

; Remember the newly requested mode

             MOVE.W             csMode(A2),saveMode(A3)    ; remember requested mode
             CMP.W        #FifthVidMode,csMode(A2)  ; is this 16/32bpp mode?
             BGE.S        @direct                   ; if not, then indexed mode
             BCLR         #DirectFlag,GFlags(A3)    ; clear the flag bit
             BRA.S        GoOn
@direct
             BSET         #DirectFlag,GFlags(A3)    ; set the flag bit
GoOn

;
; In most cards, the actual CLUT position occupied by black and white changes with
; depth changes.  This causes a number of unpleasant screen anomalies (pixels appear
; magnified when going to lower pixel depths, or colors appear when going into higher
; depths). To solve this problem, the entire CLUT is set to 50% gray while the mode
is
; changed, which masks these problems.  The SetMode call is followed by calls that
; fill the frame buffer with a 50% dithered gray pattern, then set valid CLUT
contents.
;

             BSR          GrayCLUT                  ; set the entire CLUT to 50%
gray
             BSR          HWSetDepth                ; set the depth (modeID in D1)
ModeOK1      MOVE.W             D0,savePage(A3)     ; save the new page number
             BSR          HWSetPage                 ; set the video page
                                                    ; (pageID in D0)
             MOVE.L       D0,saveBaseAddr(A3)       ; save the new base address
NoChange     MOVE.L       saveBaseAddr(A3),csBaseAddr(A2) ; return the base address
             BRA          CtlGood                   ;

             ENDWITH
```

```
SetEntries
;-----------------------------------------------------------------------
;
;  Input :    (A2) =       csTable -> table of colorSpecs (NOT colortable!)
;                          csStart -> where to start setting, or -1
;                          csCount -> # of entries to change
;
; This call has two modes.  In SEQUENCE mode, csCount entries are changed in the
CLUT, ; starting at csStart.  In INDEX mode, csCount entries are installed into the
CLUT at
; the positions specified by their value fields.  This mode is selected by passing
; csStart = -1.
;
; If the current screen depth is a direct pixel mode (16/32 bpp), then this routine
; returns an error.
;
; This code is shared with DirectSetEntries, below. Since luminance mapping should not
; occur in direct modes, the code that sets the hardware should honor the setting of
; this flag in the device-specific code.
;
; If gamma correction is implemented by table look-up, then SetEntries will pick up
; the respective red, green, and blue values, and, using the GDataWidth field from
; the gamma table, perform a look-up on each of these channel values in the gamma
; table data.
;
; This routine can optionally be implemented to execute asynchronously by posting the
; CLUT change request in a table that is loaded as part of the slot interrupt handler.
; The Macintosh will NOT call this control call with the async variant of the trap;
; rather, the SetEntries call executes as a normal control call and delays its
; hardware activity until VBL. In doing this, a few extra rules must be followed:
;
;      1) The driver should implement KillIO (see above).
;      2) If SetEntries is entered while the interrupt level is
;         nonzero, it should write immediately to the CLUT hardware.
;

             WITH         VidLocals

             MOVE.        csTable(A2),D0            ; check for a nil pointer
             BEQ          CtlBad
             BTST         #DirectFlag,GFlags(A3)    ; is this a direct video mode?
             BNE          CtlBad                    ; if so, then exit with error

; Get the gamma correction tables in registers

SECore
             MOVEM.L      A4-A6/D4-D7,-(SP)         ; save registers for gamma
             MOVE.W             GFlags(A3),D5       ; get GFlags word in D5
             MOVE.L             saveGammaPtr(A3),A0 ; get pointer to gamma data
                                                    ; structure
             MOVE.W             GFormulaSize(A0),D0 ; get the size of formula data
             LEA          GFormulaData(A0),A4       ; point to formula data
             ADD          D0,A4                     ; red correction table starts
                                                    ; here
             MOVE.L             A4,A5               ; get default pointer to green
                                                    ; data
             MOVE.L             A4,A6               ; get default pointer to blue
                                                    ; data
             MOVE         GDataWidth(A0),D7         ; get width of each entry in
bits
             CMP          #1,GChanCnt(A0)           ; if only one table, we're set
             BEQ.S        WriteCLUT                 ; only one table, so continue

             MOVE         GDataCnt(A0),D0           ; get # entries in table
             MOVE         D7,D1                     ; copy it to calculate offsets
             ADD          #7,D1                     ; round to nearest byte
             LSR          #3,D1                      ; get bytes per entry
             MULU         D1,D0                      ; get size of table in bytes

             ADDA         D0,A5                      ; calc base of green
             ADDA         D0,A6                      ; calc base of blue
             ADDA         D0,A6                      ; calc base of blue

WriteCLUT

;
;  <DEVICE-SPECIFIC>
;
; Hardware implementations vary greatly here.  Usually, based on the csStart parameter,
; the code will separately implement sequential and indexed CLUT writes. If these
; routines use substantial stack space, they should be careful to check that this
; amount of space is available.

             Bsr          HWWriteCLUT                ; left as an exercise for the
                                                     ; reader
;
; When the driver has been set to luminance map (convert from color to gray-scale
; equivalents), it should calculate the values based on a .30R/.59G/.11B ratio. If all
; output channels are being set, the gamma correction factors should still be applied.
;
; If you share this code with DirectSetEntries, then this section of code should NOT
; apply luminance mapping if DirectFlag is set.
;

             MOVEM.L      (SP)+,A4-A6/D4-D7          ; restore saved registers
             BRA.S        CtlGood                    ; exit with a good result

             ENDWITH
```

```
SetGamma
;-----------------------------------------------------------------------------------
;
;      Set the gamma table. This call copies the supplied gTable so the
;      caller does not have to put the source on the system heap.
;
;      GType in the incoming table should match the unique drHwId for this
;      card to guarantee that the table is actually intended for this device.
;      Optionally, a card may accept gamma tables with a GType of 0, if the
;      standard format is supported on this device.
;
;      If the gamma table ptr is NIL, then set the gamma table to be a linear ramp.
;       This allows the full dynamic range of the CLUT to be used.
;
;      A1 = ptr to DCE
;      A2 = ptr to cs parameter record
;      A3 = ptr to private storage
;
;-----------------------------------------------------------------------------------

             WITH         VidLocals

; Get new gamma table and check that we know how to handle it

             MOVE.L             csGTable(A2),D0     ; test for a NIL pointer
             BEQ          LinearTab                 ; if so, then set this table
                                                    ; linear
             MOVE.L             D0,A2               ; get pointer to new gamma table
             TST.W        GVersion(A2)              ; version = 0?
             BNE          CtlBad                    ; => no, return error
             TST.W        GType(A2)                 ; test the hardware ID
             BEQ.S        ChangeTable               ; if 0, we accept any gamma
table
             CMP.W        #drHwSample,GType(A2)     ; type = sample card?
             BNE          CtlBad                    ; => no, return error

; If new table is different size, reallocate memory

ChangeTable  MOVE.L       saveGammaPtr(A3),A0       ; get current gamma in A0
             MOVE         GFormulaSize(A2),D0       ; get size of formula in new
             CMP          GFormulaSize(A0),D0       ; same as current gamma table
             BNE.S        @GetNew                   ; =>no, resize pointer
             MOVE         GChanCnt(A2),D0           ; get number of tables in new
             CMP          GChanCnt(A0),D0           ; same as current gamma table?
             BEQ.S        @SizeOK                   ; => yes, data size ok
             BGT.S        @GetNew                   ; => new one is bigger,
                                                    ;     save old one
@NewSize     _DisposPtr                             ; if new one smaller, dispose
old

             CLR.L        saveGammaPtr(A3)          ; flag it's been disposed
@GetNew      MOVE         GDataCnt(A2),D0           ; get number of entries
             MULU         GChanCnt(A2),D0           ; multiply by number of tables
             ADD          GFormulaSize(A2),D0       ; add size of formula data
             ADD          #GFormulaData,D0          ; add gamma table header size
             _NewPtr      ,Sys                      ; and allocate a new pointer
             BNE          CtlBad                    ; => unable to allocate storage
             MOVE.L       saveGammaPtr(A3),D0       ; get old gamma table
             MOVE.L       A0,saveGammaPtr(A3)       ; save new gamma table
             TST.L        D0                        ; was there an old one?
             BEQ.S        @SizeOK                   ; => no, already disposed
             MOVE.L             D0,A0               ; else get old table
             _DisposPtr                             ; and dispose of old gamma table
             MOVE.L       saveGammaPtr(A3),A0       ; get new gamma table back

; Copy the gamma table header
```

```
@SizeOK      MOVE         GChanCnt(A2),D0            ; get number of tables
             MOVE         GFormulaSize(A2),D1        ; get size of formula data
             MOVE         gDataCnt(A2),D2            ; get number of entries
             MOVE.L              (A2)+,(A0)+         ; copy gamma header
             MOVE.L              (A2)+,(A0)+         ; which is
             MOVE.L              (A2)+,(A0)+         ; 12 bytes long

; Copy the data

             MULU         D0,D2                      ; multiply by number of tables
             ADD          D1,D2                      ; add in size of formula data
             SUBQ         #1,D2                      ; get count - 1
@NxtByte     MOVE.B              (A2)+,D0            ; get a byte
             MOVE.B              D0,(A0)+            ; move a byte
             DBRA         D2,@NxtByte                ; => repeat for all bytes

SGExit
             BTST         #DirectFlag,GFlags(A3)     ; is it in direct pixel mode?
             BEQ.S        OutOHere                   ;
             BSR          DirectCLUTRamps            ; put RGB channels up in
                                                     ;     direct mode
OutOHere
             BRA          CtlGood                    ; => return no error

;
; Set up a linear gamma table. To prevent memory thrash, build this new one the same
size as ; the existing one (one or three channel)
;

LinearTab
             MOVE.L       saveGammaPtr(A3),A0        ; get current gamma in A0
             MOVE.W       GFormulaSize(A0),D0        ; get size of formula in new
             MOVE.W       GChanCnt(A0),D2            ; get the number of tables
             SUBQ         #1,D2                      ; zero based, of course
             ADDA         #GFormulaData,A0           ; point to tables
             ADDA         D0,A0                      ; skip over formula data
@ChanLoop    MOVE.W              #255,D0             ; loop count within each channel
@entryLoop   NOT.B        (A0)+                      ; invert it to make table
                                                     ;     ramp properly
             DBRA         D0,@entryLoop              ; for each entry in channel
             DBRA         D2,@ChanLoop               ; and each channel
             BRA          SGExit

             ENDWITH
```

```
GrayPage
;----------------------------------------------------------------------------------
;
;     Fill the specified page in the current mode to 50% dithered gray
;
;     A1 = ptr to DCE
;     A2 = ptr to cs parameter record
;     A3 = ptr to private storage
;
;----------------------------------------------------------------------------------

             WITH         VidLocals,VDPageInfo

             MOVE         saveMode(A3),D1          ; D1 = mode
             MOVE         D1,csMode(A2)            ; force current mode for ChkPage
             BSR          ChkMode                  ; convert mode to depth in D1
             BNE          CtlBad                   ; => not a valid depth
             MOVE         csPage(A2),D0            ; D0 = page
             BSR          ChkPage                  ; check page
             BNE          CtlBad                   ; => not a valid page

             BSR          GrayScreen               ; paint the screen gray

             BTST         #DirectFlag,GFlags(A3)   ; is it in direct pixel mode?
             BEQ.S        Leave                    ;
             BSR          DirectCLUTRamps          ; put RGB channels in direct
mode
Leave
             BRA          CtlGood                  ; => return no error

             ENDWITH

SetGray
;----------------------------------------------------------------------------------
;
;   Set luminance mapping on (csMode = 1) or off (csMode = 0)
;
;   When luminance mapping is on, RGB values passed to setEntries are mapped to
;   gray-scale equivalents before they are written to the CLUT.
;
;     A1 = ptr to DCE
;     A2 = ptr to cs parameter record
;
;----------------------------------------------------------------------------------

             WITH         VidLocals

             MOVE.B             csMode(A2),D0      ; get flag value
             BFINS        D0,GFlags(A3){0:1}       ; set flag bit
             BRA          CtlGood                  ; all done

             ENDWITH
```

```
SetInterrupt
;-----------------------------------------------------------------------------------
;
;      Enable (csMode = 0) or disable (csMode = 1) VBL interrupts.
;
;      This routine enables and disables the interrupt source on the card, and
;      installs or removes the slot queue interrupt element. It doesn't
;      allocate or dispose memory.
;
;      A1 = ptr to DCE
;      A2 = ptr to cs parameter record
;      A3 = ptr to private storage
;
;-----------------------------------------------------------------------------------

             WITH         VidLocals,VDPageInfo,SlotIntQElement

             MOVE.        csMode(A2),D0                 ; get flag value
             BFINS        D0,GFlags(A3){1:1}            ; set flag bit
             BNE.S        DisableThem                   ; if zero, then enable
;
; This code enables interrupts and installs the interrupt handler
;

             BSR.S        EnableVGuts                   ; call common code
             BNE          CtlBad                        ; error, flag problem
             BRA          CtlGood                       ; and go home

;
; This code disables VBL interrupts, then removes the interrupt handler
;

DisableThem  BSR.S        DisableVGuts                  ; jump to the disabling utility
             BRA          CtlGood                       ; all done

;
; The following two routines are common code shared between the Open call and the
; SetInterrupt control call
;

DisableVGuts
             CLR          D0                            ; clear D0.W
             MOVE.B             dctlSlot(A1),D0         ; set up slot # for _SIntRemove

; <DEVICE-SPECIFIC> disable the NMRQ interrupt source here

             MOVE.L             saveSQElPtr(A3),A0  ; get the SQ element pointer
             _SIntRemove                                ; remove the interrupt handler

             RTS
```

```
EnableVGuts
            MOVE.L      saveSQElPtr(A3),A0        ; get the queue element pointer
            MOVE.W      #SIQType,SQType(A0)       ; set up queue ID
            LEA         BeginIH,A2                ; get pointer to interrupt
                                                  ; handler
            MOVE.L      A2,SQAddr(A0)             ; set up int routine address

            MOVE.L      A1,SQParm(A0)             ; this field is passed to the
                                                  ; handler, and can be any
                                                  ; convenient value
                                                  ; (here, the DCTlHandle)

            MOVE.B      dctlSlot(A1),D0           ;
            _SIntInstall                          ; and do install
            BNE.S       IntBad

; <DEVICE-SPECIFIC> enable the NMRQ interrupt source here

            RTS                                   ; return home
;
; in the event there is a problem, return Z-flag off
;

IntBad
            MOVEQ       #1,D0                     ; clear Z bit
            RTS

            ENDWITH
```

```
DirectSetEntries
;---------------------------------------------------------------------
;
;     This card allows specialized applications to change the color table hardware
;     (if present) while in direct pixel modes. It has exactly the same
;     interface as SetEntries, but does not return an error when called in
;     direct mode. If the current mode is an indexed mode, then this routine
;     returns CtlBad.
;
;     A1 = ptr to DCE
;     A2 = ptr to cs parameter record
;     A3 = ptr to private storage
;
;---------------------------------------------------------------------


            WITH        VidLocals

            BTST        #DirectFlag,GFlags(A3)     ; test if the mode is a
                                                   ;   direct one
            BEQ.S       CtlBad                     ; if not, then return an error
            BRA         SECore                     ; call the SetEntries routine

            ENDWITH

SetDefaultMode
;--------------------------------------------------------------------
;
;     Write the spID of the card's new default mode into slot PRAM. This routine
;     is used to support video mode families.  Via its monitor type sensing
;     capabilities, PrimaryInit can decide which of the video sResource lists
;     should be selected at startup.  When the new Slot Manager is present, it is
;     possible to designate inactive alternate video sRsrc lists as well as the
;     primary list.  These alternate sRsrcs appear in the Options dialog box of the
;     Monitors cdev, and allow the alternate mode to be selected as the primary
;     display mode upon reboot.  The selection of the default sRsrc list is set in
;     slot PRAM by a call to this routine.  Note that alternate sRsrcs should
;     always generate video timing that is compatible with the connected monitor.
;     Noncompatible timings should only be selected via monitor sense-line
;     detection.

;     A1 = ptr to DCE
;     A2 = ptr to cs parameter record
;     A3 = ptr to private storage
;
;--------------------------------------------------------------------

            WITH        VidLocals,spBlock,VDFlagInfo


;
; Set up a slot parameter block on the stack
;

            SUBA        #spBlockSize,SP            ; make an spBlock on the stack
            MOVE.L             SP,A0               ; get pointer to parm block now
            MOVE.B      dCtlSlot(A1),spSlot(A0)    ; put slot in pBlock
            CLR.B       spExtDev(A0)               ; external device = 0


;
; Read the slot PRAM to determine what the currently saved mode is. The first byte is
; the board ID, followed by the default screen depth. This sample keeps the default
; spID in VendorUse2.  Remember that, for video cards only, VendorUse1 is reserved for
; the system to identify the spID of the structure that contains the current screen
; depth.
;
```

```
             SUBA         #SizesPRAMRec,SP           ; allocate block for PRAM record
             MOVE.L              SP,spResult(A0)     ; point to it
             _sReadPRAMRec                           ; read it

;
; The parameter list ID (identifying the screen depth) in 2(SP) is still valid.
;
; It is very important that Monitors (or someone) invalidate and set up the screen
; resource if this call is exercised.  The information on how to set up the 'scrn'
; resource for the next boot is all available with judicious use of the new Slot
; Manager routines.  Monitors is also responsible for setting up the new default
; screen depth in PRAM.
;

             MOVE.B              csMode(A2),3(SP)    ; write the mode into PRAM
                                                     ; buffer
             MOVE.L              SP,spsPointer(A0)   ; set up parameter block
             _SPutPRAMRec                            ; write the new record out
             ADDA         #SizesPRAMRec+spBlockSize,SP ; deallocate buffer
             BRA          CtlGood

             ENDWITH

;-----------------------------------------------------------------------------------
;
; VideoClose releases the device's private storage and removes the interrupt handler.
;
;
; Entry:     A0 = param block pointer
;            A1 = DCE pointer
;
;-----------------------------------------------------------------------------------
VideoClose   WITH         VidLocals

             MOVE.        A3,-(SP)                   ; save A3
             MOVE.L              dCtlStorage(A1),A3
             MOVE.L              (A3),A3             ; get pointer to private storage

             BSR          DisableVGuts               ; deactivate interrupts

             MOVE.L       saveSQElPtr(A3),A0         ; get interrupt handler queue
                                                     ; elem
             _DisposPtr                              ; dispose it
             MOVE.L       saveGammaPtr(A3),A0        ; get pointer to gamma table
             _DisposPtr                              ; dispose it
             MOVE.L       dCtlStorage(A1),A0         ; dispose of the private storage
             _DisposHandle                           ;

             MOVEQ        #0,D0                      ; no error
             MOVE.L              (SP)+,A3            ; restore A3
             RTS                                     ; and return
             ENDWITH
```

```
;-----------------------------------------------------------------------------
;
; Video driver status call handler
;
;      (0)  Error
;      (1)  Error
;      (2)  GetMode
;      (3)  GetEntries
;      (4)  GetPage
;      (5)  GetPageBase
;      (6)  GetGray
;      (7)  GetInterrupt
;      (8)  GetGamma
;      (9)  GetDefaultMode
;
;   Entry:   A0 = param block
;            A1 = DCE pointer
;
;   Exit:    D0 = error code
;
;--------------------------------------------------------------------------

VideoStatus  MOVEM.L     A0/D1/D2,-(SP)           ; save some registers
             MOVE.W           csCode(A0),D0       ; get the selector
             MOVE.L           csParam(A0),A2      ; A2 <- ptr to control
parameters
             MOVE.L           dCtlStorage(A1),A3
             MOVE.L           (A3),A3             ; get pointer to private storage
             CMP.W       #9,D0                    ; if csCode not in [0..9] then
             BHI.S       StatBad                  ; error, csCode out of bounds

             MOVE.W           StatJumpTbl(PC,D0.W*2),D0 ; get the routine's
relative
                                                  ; offset
             JMP         StatJumpTbl(PC,D0.W)     ; jump to it

StatJumpTbl  DC.W        StatBad-StatJumpTbl      ;$00 => Error
             DC.W        StatBad-StatJumpTbl      ;$01 => Error
             DC.W        GetMode-StatJumpTbl      ;$02 => GetMode
             DC.W        GetEntries-StatJumpTbl   ;$03 => GetEntries
             DC.W        GetPage-StatJumpTbl      ;$04 => GetPage
             DC.W        GetPageBase-StatJumpTbl  ;$05 => GetPageBase
             DC.W        GetGray-StatJumpTbl      ;$06 => GetGray
             DC.W        GetInterrupt-StatJumpTbl ;$07 => GetInterrupt
             DC.W        GetGamma-StatJumpTbl     ;$08 => GetGamma
             DC.W        GetDefaultMode-StatJumpTbl;$09 => GetDefaultMode


StatBad      MOVEQ       #statusErr,D0            ; else say we don't do this one
             BRA.S       StatDone                 ; and return

StatGood     MOVEQ       #noErr,D0                ; return no error
StatDone     MOVEM.L     (SP)+,A0/D1/D2           ; restore registers
             BRA         ExitDrvr
```

```
GetMode
;-------------------------------------------------------------------------------------
;
;      Return the current mode.
;
;      Inputs :      A2 = pointer to csParams
;                    A3 = pointer to private storage
;
;-------------------------------------------------------------------------------------

             WITH         VidLocals,VDPageInfo

             MOVE.W       saveMode(A3),csMode(A2)           ; return the mode
             MOVE.W       savePage(A3),csPage(A2)           ; return the page number
             MOVE.L       saveBaseAddr(A3),csBaseAddr(A2)   ; and the base address
             BRA.S        StatGood                          ;

             ENDWITH

GetEntries
;-------------------------------------------------------------------------------------
;
;      Read the current contents of the CLUT. This routine, unlike SetEntries,
;      doesn't return an error if the device is in direct mode. No attempt
;      is made to reverse the effects of gamma table adjustment.
;
;      Inputs : A2 = pointer to csParams
;
;-------------------------------------------------------------------------------------

             WITH         VidLocals

             MOVE.        csTable(A2),D0                    ; check for a nil pointer
             BEQ.S        StatBad

;
;   <DEVICE-SPECIFIC>
;
; Hardware implementations vary greatly here. Usually, based on the csStart
parameter,
; the code should support both sequential and indexed CLUT writes. If these routines
; use substantial stack space, they should be careful to check that this amount of
; space is available.
;
             BSR          HWGetCLUT                         ; left as an exercise for
                                                            ; the reader

             BRA          StatGood                          ; => return no error

             ENDWITH
```

```
GetPage
;-------------------------------------------------------------------------------------
;
;      Return the number of pages in the specified screen depth. The number of
;      pages is always a counting number, not zero-based. Page counts are
;      only visible for the various depths in this video sResource.
;
;-------------------------------------------------------------------------------------

             WITH         VidLocals,VDPageInfo

             MOVE         csMode(A2),D1             ; get the mode
             MOVE         D1,D2                     ; keep a copy
             BSR          ChkMode                   ; is this mode OK?
             BGT          StatBad                   ; => not a valid mode
             SUB          #FirstVidMode,D2          ; mode, zero-based
             MOVE.L       saveVidParms(A3),A0       ; get pointer to vid parameters
             MOVE.W       D_Pages(A0,D2*4),D1       ; get the number of video pages
             MOVE.W       D1,csPage(A2)             ; return page count (high byte
                                                    ;   0 from ChkMode)
             ADD.W        #1,csPage(A2)             ; turn into a counting number
             BRA          StatGood                  ; => return no error

             ENDWITH

GetPageBase
;-------------------------------------------------------------------------------------
;
;      Return the base address for the specified page in the current mode.
;
;-------------------------------------------------------------------------------------

             WITH         VidLocals,VDPageInfo

             MOVE         saveMode(A3),D1           ; get the current mode
             MOVE         D1,csMode(A2)             ; force current mode, just in
                                                    ;   case for ChkPage
             BSR          ChkMode                   ; convert to depth in D1
             MOVE.W       csPage(A2),D0             ; get the requested page
             BSR          ChkPage                   ; is the page valid?
             BNE          StatBad                   ; => no, just return
             MOVE         saveMode(A3),D1           ; get the current screen depth
ID
             SUB          #OneBitMode,D1            ; make it zero based
             MOVE.L       saveVidParms(A3),A0       ; point to data table
             MULU         (D_RowBytes,A0,D1*2),D0   ; calc page * rowBytes
             MULU         D_Height(A0),D0           ; calc page * rowBytes * height
             ADD.L        #defmBaseOffset,D0        ; here's the QuickDraw offset
                                                    ;   value to be added
@2           ADD.L        dCtlDevBase(A1),D0        ; add base address for card
             MOVE.L       D0,csBaseAddr(A2)         ; return the base address
             BRA          StatGood                  ; => return no error

             ENDWITH
```

```
GetGray
;------------------------------------------------------------------------------------
;
;      Return a Boolean, set true if luminance mapping is on.
;
;------------------------------------------------------------------------------------

             WITH         VidLocals,VDFlagInfo

             BFEXTU       GFlags(A3){0:1},D0          ; get the state of flag
             MOVE.B       D0,csMode(A2)               ; return value
             BRA          StatGood                    ; => and return

             ENDWITH

GetInterrupt
;------------------------------------------------------------------------------------
;
;      Return a Boolean in csMode, set true if VBL interrupts are disabled.
;
;------------------------------------------------------------------------------------

             WITH         VidLocals,VDFlagInfo

             BFEXTU       GFlags(A3){1:1},D0          ; get the state of flag
             MOVE.B       D0,csMode(A2)               ; return value
             BRA          StatGood                    ; => and return

             ENDWITH

GetGamma
;------------------------------------------------------------------------------------
;
;      Return the pointer to the current gamma table.
;
;------------------------------------------------------------------------------------

             WITH         VidLocals

             MOVE.L       saveGammaPtr(A3),csGTable(A2)     ; return the pointer
             BRA          StatGood                          ; and return a good result

             ENDWITH
```

```
GetDefaultMode
;------------------------------------------------------------------------------------
;
;      Read the card default mode from slot PRAM.
;
;      A1 = ptr to DCE
;      A2 = ptr to cs parameter record
;      A3 = ptr to private storage
;
;------------------------------------------------------------------------------------

             WITH         spBlock,VDFlagInfo

; Set up a slot parameter block on the stack

             SUBA         #spBlockSize,SP           ; make an spBlock on stack
             MOVE.L       SP,A0                     ; get pointer to parm block now
             MOVE.B       dCtlSlot(A1),spSlot(A0)   ; put slot in pBlock
             CLR.B        spExtDev(A0)              ; external device = 0

;
; Read the slot PRAM to determine what the currently saved mode is. The first byte is
; the board ID, followed by the default mode. This sample keeps the new default video
; mode in VendorUse2, which is what is returned as the result of this routine.
;

             SUBA         #SizesPRAMRec,SP               ; allocate block for PRAM
                                                         ;   record
             MOVE.L       SP,spResult(A0)                ; point to it
             _sReadPRAMRec                               ; read it

             MOVE.B       3(SP),csMode(A2)               ; return the result
             ADDA         #SizesPRAMRec+spBlockSize,SP   ; release buffer
             BRA          StatGood

             ENDWITH

;------------------------------------------------------------------------------------
;
;      Exit from control or status.
;
;------------------------------------------------------------------------------------


ExitDrvr     BTST         #NoQueueBit,ioTrap(A0)    ; no queue bit set?
             BEQ.S        GoIODone                  ; => no, not immediate
             RTS                                    ; otherwise, it was an immediate
                                                    ;   call

GoIODone     MOVE.L       JIODone,A0                ; get the IODone address
             JMP          (A0)                      ; invoke it
```

```
;-------------------------------------------------------------------------------------
;
;      Utilities
;
;-------------------------------------------------------------------------------------

ChkMode
;-------------------------------------------------------------------------------------
;
;   Verifies that the screen depth is available in this video sResource.
;
;      ->  D1: mode
;      ->  A3: pointer to driver privates.
;
;   Returns EQ if mode is valid. All registers preserved
;
;-------------------------------------------------------------------------------------

             WITH          VidLocals

             MOVE.L        A0,-(SP)            ; save a register
             CMP.W         #FirstVidMode,D1    ; compare to lowest mode ($80)
             BMI.S         ModeBad             ; exit with bad mode
             MOVE.L        saveVidParms(A3),A0 ; point to parameters for this mode
             CMP.W         D_MaxDepthID(A3),D1 ; compare to depth range for this
                                               ;   config
             BGT.S         ModeBad             ; exit if out of range
ModeOK              CMP.W  D1,D1               ; get EQ
ModeBad      MOVE.L        (SP)+,A0            ; restore saved register (doesn't
                                               ;   affect flags)
             RTS                               ; EQ if valid depth

             ENDWITH

ChkPage
;-------------------------------------------------------------------------------------
;
;   Checks to see if the page number in D0 is valid for the depth in D1.
;
;      ->  D0: page
;      ->  D1: depth
;      ->  A3: pointer to driver privates
;
;   Returns EQ if page is valid.  All registers preserved.
;
;-------------------------------------------------------------------------------------

             WITH          VidLocals

             MOVEM.L       D2/A1,-(SP)         ; save work registers
             MOVE.W        saveMode(A3),D2     ; get offset to page data
             SUB.W         #OneBitMode,D2      ; zero-based offset in D2
             MOVE.         saveVidParms(A3),A1 ; get pointer to data tables
             CMP.W         D_Pages(A1,D2*4),D0 ; compare to zero-based page count
             SGT           D2                  ; set flag if too big
             TST.B         D2                  ; and test condition
             MOVEM.L       (SP)+,D2/A1         ; restore work registers
             RTS
             ENDWITH
```

```
HWSetDepth
;---------------------------------------------------------------------------------
;
;   This utility sets the screen depth hardware.
;
;      ->  D1: new screen depth ID (already verified)
;      ->  A1: DCE pointer
;      ->  A2: parameter block pointer
;      ->  A3: private storage pointer
;
;    Preserves all registers.
;
;---------------------------------------------------------------------------------

;
; <DEVICE-SPECIFIC>
;
; Simply convert the screen depth ID to information appropriate to performing a
; screen depth change here.
;
             RTS
HWSetPage
;---------------------------------------------------------------------------------
;
;  The base of a page is at dCtlDevBase + defmBaseOffset + (page * RowBytes * height).
;
;      ->  D0: new page number (already verified)
;      ->  D1: new screen depth ID (already verified)
;      ->  A1: DCE pointer
;      ->  A2: parameter block pointer
;      ->  A3: private storage pointer
;
;      <-  D0: return the base address
;
;---------------------------------------------------------------------------------

             WITH         VidLocals

             MOVEM.L      A0/D1,-(SP)            ; save some registers
             MOVE         saveMode(A3),D1        ; get the current
             SUB          #OneBitMode,D1         ; make it zero based
             MOVE.L       saveVidParms(A3),A0    ; point to data table
             MULU         D_RowBytes(A0,D1*4),D0 ; calc page * rowBytes
             MULU         D_Height(A0),D0        ; calc page * rowBytes * height
             ADD.L        #defmBaseOffset,D0     ; add QD offset

;
; <DEVICE-SPECIFIC>
;
; Set the screen page based on this offset information here
;

             ADD.L        dCtlDevBase(A1),D0     ; add the card base address

             MOVEM.L      (SP)+,A0/D1            ; restore all registers
             RTS                                 ; and return

             ENDWITH
```

```
GrayScreen
;----------------------------------------------------------------------------------
;
;      ->  D0: page to gray
;      ->  A3: private storage pointer
;
;  All registers are preserved.
;
;----------------------------------------------------------------------------------

             WITH         VidLocals

             MOVEM.L      D0-D7/A0-A1,-(SP)         ; save all registers
             MOVE         saveMode(A3),D1           ; get the mode
             SUB          #FirstVidMode,D1          ; make it zero based

             LEA          Pats,A1                   ; get a pointer to the pattern
                                                    ; table
             MOVE.L       (A1,D1*4),D5              ; D5 = the proper pattern
             MOVE.L       D5,D6                     ; copy it
             NOT.L        D6                        ; and invert for 32bpp
             MOVE.L       saveVidParms(A3),A1       ; point to data table
             MOVE.W       D_RowBytes(A1,D1*4),D4    ; D4 = rowbytes for the screen
             MOVE.W       D_Height(A1),D3           ; D3 = screen height

             MULU         D4,D0                     ; rowbytes*page
             MULU         D3,D0                     ; rowbytes*page*height
             MOVE.L       D0,A1                     ; get base address in A-reg
             ADDA         #defmBaseOffset,A1        ; add offset

             SUBQ         #1,D3                     ; make height zero based

             MOVE.L       A1,A0                     ; point to the start
             LSR          #2,D4                     ; get longs per row
             SUBQ         #1,D4                     ; make count zero based

             LEA          @Continue32,A0            ; make the PC 32-bit clean
             MOVE.L       A0,D0                     ; get in D0
             _StripAddress                          ;
             JMP          (A0,D0)                   ; OK with the 68020
@Continue32
             MOVEQ        #true32b,D0               ; switch to 32-bit addressing
                                                    ; mode
             _SwapMMUMode                           ; flip to 32

@NxtRow      MOVE.L       D4,D2                     ; reload rowcount each time
@NxtLong     MOVE.L       D5,(A0)+                  ; write longword pattern to
                                                    ; screen
             CMP.W        #SixthVidMode-FirstVidMode,D1 ; if we don’t have 32bp
             BNE.S        @LittlePix                ; go on
             MOVE.L       D6,(A0)+                  ; write longword pattern to
                                                    ; screen
@LittlePix   DBF          D2,@NxtLong               ; for the entire line
             NOT.L        D5                        ; invert pattern
             NOT.L        D6                        ; for 32bpp, too
             DBF          D3,@NxtRow                ; do it for all screen lines

             _SwapMMUMode                           ; flip back to previous
                                                    ; addressing
             MOVEM.L      (SP)+,D0-D7/A0-A1         ; restore all registers
             RTS                                    ; and return

Pats         DC.L         OneBitGray,TwoBitGray,FourBitGray,EightBitGray
             DC.L         SixteenBitGray,ThirtyTwoBitGray

             ENDWITH
```

```
GrayCLUT
;------------------------------------------------------------------------------------
;
; This utility fills the entire CLUT with 50% gamma-corrected gray in support of
; video mode changes.
;
; All registers are preserved.
;
;------------------------------------------------------------------------------------
;
; <DEVICE-SPECIFIC>
;
; Find the appropriate 50% gray level and load the CLUT such that all pixel values
; produce this color on the screen.
;

             RTS
DirectCLUTRamps
;------------------------------------------------------------------------------------
;
; This utility is called in direct pixel modes to fill the CLUT hardware with linear
; ramps in the R, G, and B channels.  Note that these ramps run from 0 to all ones
; ASCENDING rather than descending as is normal in indexed modes (black = {0,0,0}
; in direct modes).
;
; All registers are preserved.
;
;------------------------------------------------------------------------------------
;
; <DEVICE-SPECIFIC>
;
; Generate a linear ramp from 0 to all ones and set this ramp in each channel of the
; CLUT. These values should be gamma corrected (across all three channels) if possible.
;

             RTS
```

```
;-----------------------------------------------------------------------------------
;
;     The slot interrupt handler
;
;-----------------------------------------------------------------------------------

; On entry, A1 contains the SQParm value passed to _sIntInstall above
; (in this case the DCE handle)

BeginIH      MOVE.L       (A1),A0                   ; deref the handle

;
; <DEVICE-SPECIFIC>
;
; Clear the NMRQ interrupt here
;

             MOVEQ        #0,D0                     ; clear D0
             MOVE.B       dCtlSlot(A0),D0           ; setup the slot number
             MOVE.L       JVBLTask,A0               ; call the VBL task manager
             JSR          (A0)                      ; with slot # in D0

             MOVEQ        #1,D0                     ; signal that int was serviced
             RTS                                    ; and return to caller
```

```
;------------------------------------------------------------------------------------
;
; Data tables
;
; These tables contain information for the driver about available modes and screen
; size information.
;
;      MaxDepthID = spID of maximum screen depth supported where:
;
;             1-bit = $80 FirstVidMode
;             2-bit = $81 SecondVidMode
;             4-bit = $82 ThirdVidMode
;             8-bit = $83 FourthVidMode
;            16-bit = $84 FifthVidMode
;            32-bit = $85 SixthVidMode
;
; Note:  If your tables are very large, then you should consider making an
; sResource directory for them.  This way you won't be filling up the system
; heap unnecessarily.  The spID for your directory should be in the nonreserved
; range of your functional sRsrcs.
;------------------------------------------------------------------------------------

Small24Parms
MaxDepth     DC.W         $83                 ; maximum screen depth ID
Height              DC.W  defmBounds_Bs       ; screen height
Pages        DC.W         Pages1s             ; number of screen pages
RowBites     DC.W         RB1s                ; rowbytes for this mode
             DC.W         Pages2s,RB2s        ; pages, rowbytes
             DC.W         Pages4s,RB4s        ; pages, rowbytes
             DC.W         Pages8s,RB8s        ; pages, rowbytes

Small32Parms
             DC.W         $85                 ; maximum screen depth ID
             DC.W         defmBounds_Bs       ; screen height
             DC.W         Pages1s,RB1s        ; pages, rowbytes
             DC.W         Pages2s,RB2s        ; pages, rowbytes
             DC.W         Pages4s,RB4s        ; pages, rowbytes
             DC.W         Pages8s,RB8s        ; pages, rowbytes
             DC.W         Pages16s,RB16s      ; pages, rowbytes
             DC.W         Pages32s,RB32s      ; pages, rowbytes

BigParms
             DC.W         $83                 ; maximum screen depth ID
             DC.W         defmBounds_Bb       ; screen height
             DC.W         Pages1b,RB1b        ; pages, rowbytes
             DC.W         Pages2b,RB2b        ; pages, rowbytes
             DC.W         Pages4b,RB4b        ; pages, rowbytes
             DC.W         Pages8b,RB8b        ; pages, rowbytes

OneParms
             DC.W         $80                 ; maximum screen depth ID
             DC.W         defmBounds_Bs       ; screen height
             DC.W         Pages1s,RB1s        ; pages, rowbytes

D_MaxDepthID EQU          MaxDepth-Small24Parms
D_Height     EQU          Height-Small24Parms
D_Pages      EQU          Pages-Small24Parms
D_RowBytes   EQU          RowBites-Small24Parms
```

Appendix D **PAL Listing for the NuBus Test Card**

This is a listing of the PAL-implemented logic equations for the NuBus Test Card described in Chapter 10, "NuBus Design Examples."

```
.ident PAL16R8,B SLAVE, NuBus slave controller
       Version:   1.1

.names

              /CLK
     /START   /ACK     /MYSLOT  /RESET   /MSTDN   /TM1     A19D11   A18D10
              Gnd      /OE
     A18D10L  A19D11L  TM1L     MASTER   /romoe1  /ROMOE   /ACKCY   SLAVE
              Vcc

.equations

    /SLAVE       :=  RESET
                        {initialization}
                 +  /SLAVE * /START
                 +  /SLAVE * ACK
                 +  /SLAVE * /MYSLOT
                        {holding; DeMorgan of START * /ACK * MYSLOT}
                 +   SLAVE * ACKCY
                        {clearing term}
                 ;

    /MASTER      :=  RESET
                        {initialization}
                 +  /MASTER * /SLAVE
                 +  /MASTER * /TM1L
                 +  /MASTER * /A19D11L
                 +  /MASTER *  A18D10L
                        {holding term; DeMorgan of:
                           SLAVE * TM1L * A18D11L * /A18D10L }
                 +   MASTER * MSTDN
                        {clearing term, at end of MASTER cycle}
                 ;

    ROMOE        :=  START * /ACK * MYSLOT * /TM1 * A19D11 * A18D10
                     * /RESET
                        { latching term, when decoding a READ to us }
                 +   ROMOE * /ACKCY
                     * /RESET
                        { holding term thru access }
                 ;

romoe1           :=  ROMOE
                        { simply a delayed ROMOE for cycle timing }
     ACKCY       :=  START * /ACK * MYSLOT * TM1
                        {fast cycle for WRITES}
                 +  /ACKCY * SLAVE * /ROMOE
                        {slow cycle for non-ROM READS}
                 +  /ACKCY * ROMOE * romoe1 * /A19D11L
                        {slower cycle for ROM }
                 ;

     /TM1L       :=  RESET
                 +  /TM1 *    START * /ACK * MYSLOT
                        {setting term, during address cycle}
                 +  /TM1L * /START
                 +  /TM1L *            ACK
                 +  /TM1L *                 /MYSLOT

     /A19D11L    :=  /A19D11 * START * /ACK * MYSLOT *    /MASTER
                        {setting term, during SLAVE address cycle}
                 +  /A19D11L * SLAVE * /TM1L
                 +  /A19D11L * SLAVE *  TM1L * /A19D11L
                 +  /A19D11L * SLAVE *  TM1L *             A18D10L
                        {holding terms for SLAVE accesses}
                 +   ROMOE * romoe1
                        { timing term for ROM reads }
                 +  /A19D11  * SLAVE *  TM1L *  A19D11L * /A18D10L
                        {setting term for MASTER start}
                 +  /A19D11L * MASTER
                        {holding term for MASTER}
                 ;

     /A18D10L    := /A18D10  * START * /ACK * MYSLOT
                        {setting term, during address cycle}
                 +  /A18D10L * SLAVE * /TM1L
                 +  /A18D10L * SLAVE *  TM1L * /A19D11L
                 +  /A18D10L * SLAVE *  TM1L *             A18D10L
                        {holding terms for SLAVE accesses}
                 +  /A18D10  * SLAVE *  TM1L *  A19D11L * /A18D10L
                        {setting term for MASTER start}
                 +  /A18D10L * MASTER
                        {holding term for MASTER}
                 ;

.notes

       This version corresponds to the new pin-out for the "official"
test card. It also supports the ROM, with the ROMOE signal.

.end
```

```
.ident PAL16L8,B (ARB2), Nubus Arbitration logic

       Version:   1.1

.names

     nc1     /ARB    nc3     nc4     nc5     /ID3    /ID2    /ID1    /ID0
             gnd
     /ARB0i  /ARB0o  /ARB1   /ARB2   /ARB3   arb0oe  arb1oe  arb2oe  GRANT
             vcc

.equations

      .if[ ARB * ID3 ]
          ARB3     = VCC;

          /arb2oe  = /ID3 * ARB3
                   ;
      .if[ ARB * arb2oe * ID2 ]
          ARB2     = VCC;

          /arb1oe  = /ID3 * ARB3
                   + /ID2 * ARB2
                   ;
      .if[ ARB * arb1oe * ID1 ]
          ARB1     = VCC;

          /arb0oe  = /ID3 * ARB3
                   + /ID2 * ARB2
                   + /ID1 * ARB1
                   ;
      .if[ ARB * arb0oe * ID0 ]
          ARB0o    = VCC;

          /GRANT   = /ID3 * ARB3
                   + /ID2 * ARB2
                   + /ID1 * ARB1
                   + /ID0 * ARB0i
                   ;

.notes

      ARB is responsible for doing the NuBus arbitration logic. Upon
detecting any higher priority ARB<3:0> value, it will defer its
generation of lower ARB<3:0> bits.
      The GRANT signal must be timed externally to determine proper
NuBus constraints.
      This version uses a new technique to minimize skews.

.end
```

```
.ident PAL16R8,B MASTER2, NuBus master controller for test card.

      Version:   1.3

.names

             /CLK
     MASTER  GRANT   /RQST   /START  /ACK    MASTERD /RESET  A17D9
             gnd     /oe
     A17D9L  /LOCKED /arbdn  /busy   /OWNER  /DTACY  /ADRCY  /ARBCY
             fcc

.equations

    ARBCY       := MASTER * MASTERD * /OWNER * /ARCBY * /ADRCY * /DTACY *
                    /RQST
                        {wait for RQST* unsserted, while idle}
                +  MASTER * ARBCY   *  /OWNER
                 * /RESET
                        {non-locking, hold for START*}
                +  MASTER * ARBCY   *   LOCKED
                 * /RESET
                        {holding for locked access}
                ;

    ADRCY       := /A17D9L * /OWNER * ARBCY * arbdn * GRANT * /busy * /START
                +  /A17D9L * /OWNER * ARBCY * arbdn * GRANT *  busy *  ACK
                        {START* if not locking}
                +         OWNER * LOCKED * /ADRCY * /DTACY
                 * MASTER * /RESET
                        {START* for locking case, after LOCK-ATTN}
                ;

    DTACY       :=  ADRCY
                        {assert after START*}
                +   DTACY * /ACK
                 * MASTER * /RESET
                        {hold until ACK*}
                ;

    OWNER       :=  ARBCY * arbdn * GRANT * /busy * /START
                +   ARBCY * arbdn * GRANT *  busy *  ACK
                        {when bus is free, we own it next}
                +   OWNER *  ADRCY
                 * MASTER * /RESET
                        {hold before DTACY}
                +   OWNER *  DTACY * /ACK
                 * MASTER * /RESET
                        {non-locking, wait until ACK*}
                +   OWNER * LOCKED
                 * MASTER * /RESET
                               {for LOCKing case, hold fur NULL-ATTN}
                ;
    busy            := /busy * START * /ACK
                            {beginning of transaction}
                    +   busy *         /ACK
                     * /RESET
                            {hold during cycle}
                    ;

    arbdn           :=  ARBCY */START
                            {when arbitrating, force delay}
                    ;

    LOCKED          :=  A17D9L * ARBCY * arbdn * GRANT * /busy * /START
                    +   A17D9L * ARBCY * arbdn * GRANT *  busy *  ACK
                            {set for LOCK-ATN}
                    +   LOCKED * /DTACY
                     * MASTER * /RESET
                    +   LOCKED *  DTACY * /ACK
                     * MASTER * /RESET
                            {clear on NULL-ATN}
                    ;

    /A17D9L         ;= /A17D9  * /MASTER
                            {latching term}
                    +  /A17D9L *  MASTER
                            {holding term}
                    +   LOCKED
                            {clearing term, prevent another ADRCY}
                    ;

.notes

    This version is for new pin-out of the "official" test card. MasterA
handles the delayed feature of the card. Version 1.1 also fixes the timing for
arbitration.
    This version is designed to work with the new ARB2 arbitration PAL, which
has a different sense for GRANT. It also fixes a minor timing overhang on
DTACY for 2-cycle transactions.
    Version 1.3 fixes 2-cycle write by only allowing ADRCLY for 1 clock; we
originally had overlap to try to eliminate decoding glitches.

.end
```

```
.ident PAL16L8,B MISC2, local bus/transceiver controls.

      Version:   1.2

.names

        CLK      SLAVE   TM1L    A19D11L  A18D10L /ARBCY  ADRCY    DTACY   ROMOE
                 gnd
        MASTER   GAB210  /GBA    CAB      /DOE    /AOE    /DCLK    /ACLK   GAB3
                 vcc

.equations

    GBA          =  SLAVE * /TM1L
                        {SLAVE read of card}
                 +  MASTER * ADRCY
                        {MASTER address cycle}
                 +  MASTER * DTACY * A19D11L{TM1}
                        {MASTER data cycle, when writing}
                 ;

    /CAB         =  SLAVE + /CLK
                        { DeMorgan of:  /SLAVE * CLK }
                 ;

    /GAB3        =  SLAVE * /TM1L
                        {any SLAVE read}
                 +  MASTER * /ADRCY * /DTACY
                        {MASTER loading address}
                 +  MASTER * A19D11L{TM1}
                        {MASTER write}
                 ;

    /GAB210      =  SLAVE * /TM1L * /ROMOE
                        {SLAVE, non-ROM, read}
                 +  MASTER * /ADRCY * /DTACY
                        {MASTER loading address}
                 +  MASTER * A19D11L{TM1}
                        {MASTER write}
                 ;

    ACLK         =  SLAVE * CLK *  TM1L * /A19D11L * /A18D10L
                     * /ROMOE
                        {SLAVE write to address reg}
                 ;

    AOE          =  SLAVE *       /TM1L * /A19D11L * /A18D10L
                     * /ROMOE
                        {SLAVE read of address reg}
                 +  MASTER * /ADRCY * /DTACY
                        {MASTER address cycle}
                 ;
    DCLK          =  SLAVE * CLK *  TM1L * /A19D11L *  A18D10L
                       * /ROMOE
                            {SLAVE write to data reg}
                  +  MASTER *  DTACY *      /A19D11L{/TM1} *    CLK
                            {MASTER read}
                  ;

    DOE           = SLAVE *      /TM1L * /A19D11L * A18D10L
                     * /ROMOE
                            {SLAVE read of data reg}
                  +  MASTER * DTACY *     A19D11L{TM1}
                            {MASTER write data}
                  ;


.notes

    This version of PAL corresponds to the "official" NuBus test card.
Version 1.2 reflects non-overlap of ADRCY with DTACY, which fixes problem
with 2-cycle writes;

.end
```

```
.ident PAL16L8,B       NBDRVR2, NuBus bus driver.

      Version:   1.3

.names

     /ACKCY  /ARBCY  /ADRCY  /DTACY   /OWNER   /LOCKED nc7   A19D11L A18D10L
             Gnd
     nc11    /TM0    /TM1    /tmoe    /MSTDN   /rqstoe /ACK  /START  /RQST
             Vcc
.equations

    rqstoe       =  ARBCY * /ADRCY
                                {hold until START* for normal case}
                 +  ARBCY * LOCKED
                                {hold until NULL-ATTN for locked case}
                 ;

 .if[ rqstoe ]
    RQST         =  Vcc;

 .if[ OWNER ]
    START        =  /DTACY
                                {START* for all non-DTA cycles}
                 ;

    tmoe         =  ACKCY
                                {SLAVE response}
                 +  OWNER * /ADRCY
                                {for NULL-ATTN, LOCK-ATTN}
                 ;

 .if[ tmoe ]
    ACK          =  ACKCY
                                {SLAVE response}
                 +  OWNER * /ADRCY
                                {for NULL-ATTN, LOCK-ATTN}
                 ;

 .if[ tmoe ]
    TM1          =  ACKCY
                                {SLAVE response}
                 +  OWNER *  ADRCY * A19D11L
                                {START* at address cycle}
                 +  OWNER * /ADRCY * /LOCKED
                                {set for NULL-ATTN}
                 ;
 .if[ tmoe ]
    TM0          =  ACKCY
                              {SLAVE response}
                 +  OWNER *  ADRCY * A18D10L
                              {START* at address cycle}
                 +  OWNER * /ADRCY
                              {always set for xxxx-ATTN cycles}
                 ;

    MSTDN        =  OWNEr * /LOCKED *                   DTACY * ACK
                              {done at tail end of normal cycle}
                 +  OWNER * /LOCKED * ARBCY * /ADRCY * /DTACY
                              {done for locked cases}
                 ;

.notes

    This version corresponds to the "official" test card.
    Note: due to overlap of states, RQST* is held one state too long at
end of a LOCKED transaction. However, this causes no real problem. If we
are the last winner of a RQST set, then the only result is that new RQST-
ers are held off by one CLK. If there is another RQST-er left in our set,
then it will still be driving RQST. It will properly arbitrate due to the
NULL-ATTN and become the next winner. Thus, in either case, nothing bad
happens.
    Version 1.3 reflects change to ADRCY, which is now held low only
during the address cycle of a transaction.

.end
```

Appendix E PAL Listing for the SCSI-NuBus Test Card

This is a listing of the PAL-implemented logic equations for the SCSI NuBus Test Card described in Chapter 10, "NuBus Design Examples."

```
.ident PAL16R8,B stNUBUS1, control for NuBus SCSI test card.

       Version:   1.1

.names

             clk
    /START   /ACK    /mySLOT  /mySUPER  /TM1    nc7     nc8      /RESERT
             gnd     /oe
    nc12     nc13    nc14     /S2       /S1     /SUPER  /SLOT    /IOR
             vcc

.equations

    IOR          :=  START * /ACK * mySLOT   * /TM1   * /RESET
                 +   START * /ACK * mySUPER * /TM1   * /RESET
                             { set on READ to our SLOT }
                 +   IOR * /S2                        * /RESET
                             { hold until end of transaction }
                 ;

    SLOT         :=  START * /ACK * mySLOT            * /RESET
                             { select when access to us }
                 +   SLOT * /S2                       * /RESET
                             { hold thruout cycle }
                 ;

    SUPER        :=  START * /ACK * mySUPER           * /RESET
                             { select when access to us }
                 +   SUPER * /S2                      * /RESET
                             { hold thruout cycle }
                 ;
    S1           :=  SLOT  * /S1                      * /RESET
                 +   SUPER * /S1                      * /RESET
                 +   S1 * /S2                         * /RESET
                 ;

    S2           :=  S1 * /S2                         * /RESET
                 ;

.notes

    stNUBUS1 is the main control circuit of the NuBus SCSI card. This
version will decode both a SuperSlot and a normal Slot access. Notice
that all bus transactions take the same time to simplify the logic.

.end
```

```
.ident PAL16L8,B  stNUBUS2, control for NuBus SCSI test card.

        Version:   1.0

.names

     /CLK     nc2     /SLOT   /SUPER  /S1     /S2     nc7     DRQ     IRQ
              gnd
     /INTENB  /NMRQ   nc13    /ackoe  /TM0    /TM1    /ACK    DCLK    ACLK
              vcc

.equations

      /ACLK         =  SLOT
                    +  SUPER
                    +  /CLK
                          { DeMorgan of CLK * /(SLOT + SUPER) }
                    ;

      /DCLK         =  /S2
                          { clock in data on edge of ACK* }
                    ;

      ackoe         =  SLOT
                    +  SUPER
                          { try to pull ACK* up before undriving }
                    ;

 .if[ ackoe ]
      ACK           =  S2
                    ;

 .if[ ackoe ]
      TM1           =  S2
                    ;

 .if[ ackoe ]
      TM0           =  S2
                    ;
                          { assert ACK*, TM1*, TM0* during last state of cycle }

 .if[ INTENB ]
      NMRQ          =  IRQ
                    +  DRQ
                          { drive NMRQ if either is ready }
                    ;

.notes

      This PAL is driven by stNUBUS1 (which provides decoding and
timing); it generates the control signals used by the NuBus interface.

.end
```

```
.ident PAL16L8,B  stMISC, control for NuBus SCSI test card.

      Version:   1.0

.names

     /SLOT   /SUPER  /S1    /S2     A19   A18  A9    TM1L    nc9
             gnd     /oe
     /RESET  /IOW    nc13   nc14    /INTENB /RAMCS  /ROMCS   /DACK  /SCSI
             Vcc

.equations

    SCSI         =  SLOT * /A19 * /A18 * /A9          * /RESET
                 ;
    DACK         =  SLOT * /A19 * /A18 *  A9          * /RESET
                 ;
    ROMCS        =  SLOT *  A19 *  A18                * /RESET
                 ;
    RAMCS        =  SUPER                             * /RESET
                 ;
    INTENB       =  SLOT *  A19 * /A18 *  A9          * /RESET
                 +  INTENB * /SLOT                    * /RESET
                 +  INTENB * /A19                     * /RESET
                 +  INTENB *  A18                     * /RESET
                 +  INTENB *  A9                      * /RESET
                 ;
    IOW          =  SLOT  *  TM1L * /S2
                 +  SUPER *  TM1L * /S2
                 ;

.notes

    This PAL actually generates the selects and R/W strobes to the chips
on the ScSI test card. stNUBUS1 does the basic slot decoding and cycle
timing. We simply drive the signals based upon its information.
    Note that we create our own latch for INTENB. S2 behaves like the
strobe signal; the addresses will stay around after S2 goes away.

.end
```

Glossary

ack cycle: See **acknowledge cycle.**

acknowledge cycle: Last cycle of a NuBus transaction during which /ACK is asserted by a slave responding to a master. Often shortened to *ack cycle.*

active: See **asserted.**

address: A number used to identify a location in the computer's address space. Some locations are allocated to memory, others to I/O devices.

address bus: The path along which the addresses of specific memory locations are transmitted. The width of the path determines how many addresses can be accessed (addressed) directly by the computer. For an *n*-bit-wide address bus, the computer can make use of 2^n locations in memory where information can be stored. In the Macintosh II, for example, the 32-bit address bus permits the processor to access 2^{32} (4.3 billion) addresses. This is more than 250 times as many addresses as computers with a 24-bit bus (or the Macintosh II in 24-bit mode) can access (2^{24} = 16 million).

address mapping: The assignment of portions of the address space of the computer to specific devices.

address space: A range of accessible memory. See also **address mapping.**

aliasing: The act of gaining access to a memory location from several different addresses. This usually occurs in computing systems when an incomplete address decoding mechanism is used. For example, on the map of physical addresses for the Macintosh II, there are 1024 (2^{10}) different addresses (aliases) that access the same ROM location.

AMU (Address Management Unit): The Apple custom integrated circuit in the Macintosh II that performs 24-to-32-bit address mapping. It can be replaced by the optional Paged Memory Management Unit (PMMU).

arbitration contest: The mechanism used to choose which of two or more cards requesting control of the bus will become the next bus master. For Macintosh computers with NuBus, the arbitration contest requires two bus periods (at 100 μs each).

asserted: Indicates that a signal is active or true, independent of whether that logical condition is represented by a high or low voltage.

assertion edge: The clock edge on which assertion of synchronous signals takes place.

attention cycle: The name given to a particular kind of start cycle, one in which both /START and /ACK are asserted. There are two types: attention-resource-lock and attention-null cycles.

attention-null: An attention cycle that indicates the new owner of the bus does not wish to transfer data and reinstate the bus for arbitration. It also indicates the end of a data transfer using a locked resource.

attention-resource-lock cycle: An attention cycle that initiates a sequence of locked transactions that constitute a locked tenure of the current bus master. During this tenure, cards with lockable multiport resources lock them against access by local processors other than the NuBus master.

block transfer: A transaction that consists of a start cycle, multiple data cycles from or to sequential address locations, and an acknowledge cycle. The number of data cycles is controlled by the bus master and is communicated during the start cycle. See also **1X block transfer, 2X block transfer.**

board: A printed circuit board that is a built-in (permanent) part of the computer. Compare **expansion card.**

board sResource: A unique sResource in an expansion card's declaration ROM that describes the card so that the Slot Manager can identify it. An expansion card can have only one board sResource. The board sResource entries include the card's identification number, vendor information, board flags, initialization code, and so on.

boot: Another way to say *start up.* A computer boots by loading a program into memory from an external storage medium such as a disk. Starting up is often accomplished by first loading a small program, which then reads a larger program into memory. The program is said to "pull itself up by its own bootstraps"—hence the term *bootstrapping* or *booting.*

bus: A path along which information is transmitted electronically within a computer. Buses connect short-distance networks of computer devices, such as processors, expansion cards, and physical RAM; information that travels along the bus is transmitted according to a set of rules known as a *protocol.*

bus driver: The power output transistor and circuitry used to drive the input impedance of the bus, including the parallel loads of cards connected to the bus.

bus interface logic: The electronics connecting the microprocessor bus to the NuBus in the Macintosh computers.

bus locking: A mechanism for providing continuing tenure (bus ownership) by a single card. The extended tenure may include multiple transactions or attention cycles. One type of attention cycle is called a *resource lock;* therefore a bus lock may or may not include a resource lock.

bus specification: Describes the physical characteristics of the bus and the protocol that governs the use of the bus. For example, the NuBus specification defines the clock rate of the bus, the width of the bus (in bits), the maximum rate of information transfer, and so on. It also defines the protocol, or the set of commands used to transfer information among the devices using the bus.

busy: The state of the bus between start and acknowledge cycles.

byte lane: Any of 4 bytes that make up the NuBus data width. NuBus expansion cards may use any or all of the byte lanes to communicate with each other or with the Macintosh computer.

byte smearing: A feature of MC68020 and MC68030 processors that causes the data for byte and word transfers to be replicated, or smeared, across all 32 data lines.

byte swapping: The process by which the order of bytes in each 4-byte NuBus word is changed to conform to the byte order of certain processors.

cache coherence: A property of a NuBus module to maintain a cache in which data and tags reside. These data and tags are used to determine the ownership of data between multiple processors and to maintain coherence, or agreement, between data shared by the processors and memory.

cache line: Blocks of data transferred by cache memories.

cache memory: A feature that allows frequently used data to be stored in a special buffer area (cache) and accessed by logical addresses. Cache memory improves overall performance by increasing the availability of the bus to external devices without degrading the performance of the microprocessor.

cache snooping: See **snooping.**

card: See **expansion card.**

card-generic driver: A driver that is designed to work with a variety of plug-in cards.

card-specific driver: A driver that is designed to work with a single model of plug-in card.

color look-up table (CLUT): A device that converts pixel data from a video frame buffer into red, green, and blue video signals. The CLUT in the Macintosh II Video Card supports up to 256 simultaneous colors from a possible 16.8 million colors.

Color QuickDraw: An extension of the previous 8-bit color model that supports up to 32 bits per pixel on certain Macintosh models and allows these models to process and display full-color documents, images, and visual effects with startling color clarity. See also **32-bit QuickDraw.**

configuration ROM: See **declaration ROM.**

coprocessor: An auxiliary processor that is designed to relieve the demand on the main processor by performing a few specific tasks.

Generally, closely coupled coprocessors such as the MC68881 in the Macintosh II or the MC68882 in the Macintosh IIx and the Macintosh IIcx handle tasks that could be performed by the main processor running appropriate software, but that would be performed much more slowly that way. Coprocessor architectures usually favor a certain set of operations, like floating-point calculations for graphics instruction looping, and therefore they can optimize the speed at which such operations are processed.

A microprocessor on an expansion card can also function as a coprocessor to perform tasks such as running alternative operating systems.

copyback: A NuBus transaction performed by a master to write modified cache lines to memory. It is typically used to free up cache lines to service a read or write miss or flush data into an I/O buffer. The copyback function can also be used during context switching.

copyback cache: A cache that does not propagate all write cycles to memory, but holds the data in the cache until the cache line holding the data must be reused.

CPU (central processing unit): See **processor.**

cycle: For a Macintosh computer with the NuBus interface: one period of the NuBus clock, nominally 100 ns in duration and beginning at the rising edge. For a Macintosh computer with the processor-direct slot interface: one period of the processor bus clock.

data bus: The path along which general information is transmitted within the computer. The wider the data bus, the more information can be transmitted at once. The Macintosh computers that use the MC68000 processor have 16-bit data buses. The Macintosh computers that use the MC68020 and MC68030 processors have 32-bit data buses. Thus, 32 bits of information can be transferred at a time, so that information is transferred twice as fast as in 16-bit computers (assuming equal system clock rates).

data caching: A feature of the MC68030 microprocessor that allows frequently used data to be stored in a special buffer area (cache) and accessed by logical addresses. Data caching improves overall performance by increasing the availability of the bus to external devices (in systems with more than one bus master, such as a processor and a DMA device) without degrading the performance of the microprocessor.

data cycle: Any cycle in which data is known to be valid and acknowledged. It includes acknowledge cycles as well as intermediate data cycles within a block transfer.

declaration ROM: A ROM on a NuBus expansion card that contains information identifying the card and its functions, and that may also contain code or other data. Proper configuration of the declaration ROM firmware will allow the card to communicate with the computer through the Slot Manager routines.

DIP switches: Multiple single- or double-throw switches in a dual in-line package.

direct video device: A video card whose pixel value, when placed in the frame buffer controller, directly implies the color on the display screen without indexing a color look-up table (CLUT). It will support screen depths of 16 and 32 bits per pixel. Compare **indexed video device.**

DMA: Direct memory access. A technique for transferring large amounts of data into or out of memory without using the CPU.

drive: The action of a card when it causes a bus signal line to be in a known, determinate state.

driver-supported cards: Cards that are accessed indirectly via a software driver.

driving edge: The rising edge (low to high) of the central system clock (/CLK).

EDisks (electronic disks): Electronic disks that appear to the user to be very fast, silent disk drives but that use ROM or RAM for their storage media rather than floppy or hard disks. The ROM expansion card in a Macintosh Portable can function as one or more EDisks.

expansion card: A removable printed circuit card that plugs into a connector (slot) in the computer's expansion interface and allows access to the computer's microprocessor bus. For example, the NuBus expansion interface of a Macintosh II accommodates up to six NuBus expansion cards. The processor-direct slot expansion interface of a Macintosh SE/30 or a Macintosh SE accommodates only one PDS expansion card. Expansion cards are also referred to as *slot cards* or simply as *cards*. Compare **board.**

firmware: Programs permanently stored in ROM.

fixed video device: A video device that has a color look-up table (CLUT) that cannot be modified. Compare **indexed video device.**

format block: An element in a declaration ROM's firmware structure that provides a standard entry point for other elements in the structure. The format block allows the Slot Manager to find the declaration ROM and validate it.

FPU: Abbreviation for *floating-point unit.*

frame buffer: A buffer memory that stores all the picture elements (pixels) of a frame of video information.

Frame Buffer Controller (FBC): A register-controlled CMOS gate array used to generate and control video data and timing signals.

frequency modulation (FM): An IBM 3740–compatible, single-density, disk-recording format. Compare **modified frequency modulation (MFM).**

functional sResource: An sResource in an expansion card's declaration ROM that describes a specific function of the card, for example, a video sResource.

gamma correction: A function performed by the video driver of each display device configured in the system that linearizes the differences in color (or gray-scale) response. This is required because applications cannot recognize different display screens, and cannot perform screen-by-screen corrections.

gamma table: A table that compensates for nonlinearities in a monitor's color response.

geographical addressing: A method of identifying the physical location of a card on the NuBus by having four pins of each connector electrically wired to provide a one-of-sixteen code to each slot connector ($9 through $E for the Macintosh II, Macintosh IIx, and Macintosh IIfx; $A through $E for the Macintosh Quadra 900; $9 through $B for the Macintosh IIcx; $C through $E for the Macintosh IIci; $D through $E for the Macintosh Quadra 700; and $9 for the Macintosh IIsi). A card inserted into a slot connector then has the code for that slot applied to its /ID3–/ID0 lines, without any manual setting of configuration switches as required in some bus systems.

GLU: Acronym for *general logic unit,* a class of custom integrated circuits used as interfaces between different parts of the computer.

halfword: An element of information half the length of a 32-bit NuBus or microprocessor word, therefore 16 bits long. A halfword for a 16-bit microprocessor word (from an MC68000 microprocessor, for example) is 8 bits long.

heap: The area of memory in which space is dynamically allocated and released on demand using the Memory Manager.

high: For an active-low signal, synonymous with *inactive, deasserted, unasserted, false,* and *released.*

inactive: For an active-low signal, synonymous with *high, deasserted, unasserted, false,* and *released.*

indexed video device: A video card whose frame buffer controller output indexes a color look-up table (CLUT) to define a potential color. Indexed video devices support screen depths of 1, 2, 4, and 8 bits per pixel. Compare **direct video device.**

intelligent card: A card containing one or more processors that can work independently from the main processor of the computer. Intelligent cards can serve as a medium for introducing new processor technologies into a system, but most personal computer bus architectures require too much support from the main processor for this to happen. NuBus, however, is a notable exception, because it was designed specifically to support multiple processors and, hence, intelligent cards.

longword: As used in Part II of this book, an element of information consisting of 32 bits (two 16-bit words).

low: For an active-low signal, synonymous with *asserted.*

main logic board: The primary circuit board in a computer that holds the CPU, RAM, ROM, and other integrated circuits that perform the built-in logic functions of the computer. Compare **expansion card.**

master: A card that initiates the addressing of another card or the processor on the main logic board. The card addressed is at that time acting as a slave.

minor slot space: An Apple-specific term that describes the first megabye of the 16 MB standard slot space.

modified frequency modulation (MFM): An IBM System 34–compatible, double-density, disk recording format. Compare **frequency modulation (FM).**

modulo: The integer *N* measured *modulo 4* will be the remainder (0, 1, 2, or 3) from division of *N* by 4.

multiplex: To encode information so that fewer wires are needed to transmit it, and the same cable wires and connector pins can transmit different kinds of information. The NuBus multiplexes information so that 32-bit address and data communication can be performed using a single 96-pin connector and still have adequate pins available for other necessary functions. Specifically, 32 pins are used to transmit a memory address and the same 32 pins (at a different time) to transmit data.

NuBus: A 32-bit-wide synchronous, multislot expansion bus used for interfacing expansion cards to some Macintosh computers. See also **bus interface logic, NuBus expansion slot.**

NuBus expansion slot: A connector on the NuBus in a Macintosh computer into which an expansion card can be installed. The Macintosh II, Macintosh IIx, and Macintosh IIfx have six NuBus expansion slots; the Macintosh Quadra 900 has five; the Macintosh IIcx and the Macintosh IIci have three; the Macintosh Quadra 700 has two; and the Macintosh IIsi has one.

null cycle: A type of attention cycle that reinitiates bus arbitration.

1X block transfer: A block transfer in which NuBus words are transferred at a 10 MHz rate.

open collector: A bus driver that drives a line low or doesn't drive it at all.

Paged Memory Management Unit (PMMU): The Motorola MC68851 chip, used in the Macintosh II computer to perform logical-to-physical address translation and paged memory management for virtual-memory operating systems such as A/UX. The PMMU can be installed as an option, replacing the AMU.

PAL: An integrated circuit implementing programmable array logic.

parked: A NuBus master that has released /RQST is said to be parked on the bus until another card asserts /RQST.

PDS: See **processor-direct slot (PDS).**

peer cards: Cards that are designed to execute code that is not specialized to the card—for example, two cards that are executing cooperating processes to solve a problem.

period: The 100 ns duration of the NuBus /CLK signal consisting of a 75 ns high state and a 25 ns low state.

PIO (programmed input/output): An interfacing technique where the processor directly accesses registers assigned to I/O devices by executing processor instructions. Memory-mapped I/O port registers are addressed as memory locations.

primary initialization code: A special code in an expansion card's declaration ROM that when executed performs key, one-time initialization of the card.

processor: Same as CPU, where the term *central* processing unit may not be literally applicable. The processor contains an arithmetic and logic unit (ALU) and system control hardware. In Macintosh systems containing expansion cards, there may be two or more processors (or CPUs), with none being more central in function than the others; these are multiprocessor systems.

processor-direct slot (PDS): The expansion interface architecture used on compact, or small-footprint, Macintosh computers such as the Macintosh SE and the Macintosh SE/30. It has a single connector that allows an expansion card direct access to all of the microprocessor signals.

pseudoslot design: The recommended method of designing a 68030 Direct Slot expansion card to occupy an address location that corresponds to the 32-bit physical address ranges used by NuBus expansion cards in Macintosh computers.

QuickDraw: The part of the Macintosh Toolbox that performs all graphics operations on the screen.

read hit: An operation in which a processor successfully performs an initial read of data in the cache. A read hit occurs when the requested location within the memory cache is valid.

read miss: An operation in which a processor fails to perform an initial read of data in the cache. A read miss occurs when the requested location within the memory cache is invalid.

release: To do the opposite of **drive** to a signal line.

released: For an active-low signal, synonymous with *high, inactive, deasserted, unasserted,* and *false.*

resource locking: The action of a local processor operating in a multiprocessor environment to lock the bus from NuBus intrusion while using a resource that is accessible by both the local processor and the NuBus.

sampling edge: The falling edge (high to low) of the central system clock.

scaled pixel clock period: A normalizing parameter used in the description of video card operation. One scaled pixel clock period equals 16 times the ratio of pixel clock period to the pixel depth (in bits per pixel).

SCSI (Small Computer System Interface): An industry standard parallel bus that provides a consistent method of connecting computers and peripherals.

SCSI devices: Devices, such as hard disks and tape backup units, that use the Small Computer System Interface.

68000 Direct Slot: The 96-pin expansion interface connector used on Macintosh SE and Macintosh Portable computers to allow an expansion card direct access to the MC68000 microprocessor. The connectors are physically identical but electrically different. See also **processor-direct slot (PDS).**

68020 Direct Slot: The 96-pin expansion interface connector used on Macintosh LC computers to allow an expansion card direct access to the MC68020 microprocessor. See also **processor-direct slot (PDS).**

68030 Direct Slot: The 120-pin expansion interface connector used on Macintosh SE/30 and Macintosh IIfx computers to allow an expansion card direct access to the MC68030 microprocessor. The connectors are physically identical but electrically different. See also **processor-direct slot (PDS).**

68040 Direct Slot: The 140-pin expansion interface connector used on the Macintosh Quadra 700 and Macintosh Quadra 900 computers to allow an expansion card direct access to the MC68040 microprocessor. The connectors are physically and electrically identical. See also **processor-direct slot (PDS).**

slave: A NuBus card that responds to being addressed by another card acting as a master. The Macintosh main logic board may be either master or slave. Some cards may be slave-only in function because they lack the circuitry to arbitrate in a bus ownership contest.

sleep state: The time when the Macintosh Portable is not in use and most of the circuits are powered down, the screen is blank, and the hard disk stops spinning. This state extends battery life by reducing power consumption to almost nothing.

slot: (1) A connector attached to the processor bus or the NuBus. A card may be inserted into any of the physical slots when more than one is provided (a Macintosh computer with NuBus provides from one to six slots). (2) A region in address space **(standard slot space)** allocated to a physical slot.

slot card: See **expansion card.**

slot ID: The hexadecimal number corresponding to each card slot. For the Macintosh computers with NuBus, each slot ID is established by the main logic board of the computer and communicated to the card through the /IDx lines.

Slot Manager: A set of Macintosh computer ROM routines that communicate with an expansion card's declaration ROM and allow an application to gain access to declaration ROM.

slot resource: See **sResource (slot resource).**

slot space: The address space assigned to NuBus expansion cards and PDS expansion cards that emulate NuBus cards in Macintosh computers. See also **standard slot space, super slot space.**

snarf: An action taken by a cache-coherent master when it eavesdrops on a write-back transaction and absorbs the data.

snooping: A hardware function that allows the cache to monitor bus activity by alternate bus masters.

snooping module: A module that snoops a cache-coherent transaction between a NuBus master and slave. See also **snooping.**

sResource (slot resource): An element in the firmware structure of an expansion card's declaration ROM that defines a function or capability of the card. The small *s* indicates a slot resource as opposed to a standard Macintosh resource. There is one functional sResource for each function a card can perform, but only one board sResource that identifies the card.

sResource directory: An element in a declaration ROM's firmware structure that lists all the sResources and provides an offset to each one.

`sRsrcType`: A required entry in every sResource, whose fields are used by the Slot Manager to identify the expansion card and the function it performs. An `sRsrcType` entry contains four major fields (category, type, driver hardware, and driver software), which are structured in hierarchical order.

stack: The area of memory in which space is allocated and released in LIFO (last in, first out) order.

standard slot space: The upper one-sixteenth of the total address space. These addresses are in the form $Fsxx xxxx, where *F, s,* and *x* are hex digits of 4 bits each. This address space is geographically divided among the NuBus slots according to slot ID number. Compare **super slot space.**

start cycle: The first period of a transaction during which /START is asserted. The start cycle is one bus clock period long; the address and transfer type are valid during this cycle.

state machine: A block of logic, implemented in hardware or software, that can assume a finite number of values or states, and that makes a translation from one state to another in a set sequence in response to specific inputs. For each state, a state machine generates a specific output, or asserts or deasserts a specific signal.

super slot space: The large portion of memory in the range $9000 0000 through $EFFF FFFF. NuBus addresses of the form $sxxx xxxx (that is, $s000 0000 through $sFFF FFFF) address the super slot space that belongs to the card in slot s, where *s* is an ID digit in the range $9 through $E. Compare **standard slot space.**

tenure: A period of unbroken bus ownership by a single master. A master may lock the bus and, during one tenure, perform several transactions.

32-bit QuickDraw: A system extension to handle direct color, as opposed to indexed color. In System 7, the features of 32-bit QuickDraw are included in Color QuickDraw. See also **Color QuickDraw.**

time-out period: The time period that a bus master waits for a nonresponding slave to respond before generating a bus time-out error code.

transaction: A complete NuBus operation such as a read or write. In the Macintosh computers with NuBus, a transaction is made up of an address cycle, wait cycles as required by the responding card, and a data cycle. Address cycles are one clock period long and convey address and command information. Data cycles are also one clock period long and convey data and acknowledgment information.

transfer mode: One of the 16 modes or encodings that specify which part of the addressed 32-bit word is to be transferred.

tristate: A bus driver that drives a line low or high or that doesn't drive it at all.

2X block transfer: A block transfer in which NuBus words are transferred at a 20 MHz rate.

unasserted: For an active-low signal, synonymous with *high, deasserted, false, inactive,* and *released.*

wired-OR: The physical connection of two or more input signal wires to provide a logical OR operation. If one or more of the input signals are true, the output is true. The output is false only when all of the input signals are false.

word: An element of information. As used in Part I of this book, a NuBus word is 32 bits long, a NuBus halfword, 16 bits. As used in Part II of this book, an MC68000, MC68020, MC68030, and MC68040 word is 16 bits long; a longword, 32 bits long.

write-back: An action taken by a NuBus snooping module when it returns, or writes back, its modified data to shared memory.

write-back cache: A cache that does not propagate all write cycles to memory, but holds the data in the cache until the cache line holding the data must be reused. A write-back cache, or copyback cache, is used in the NuBus cache-coherency protocol.

write hit: An operation in which a processor successfully performs an initial write of data to the cache. A write hit occurs when the requested location within the memory cache is invalid.

write miss: An operation in which a processor fails to perform an initial write of data to the cache. A write miss occurs when the requested location within the memory cache is invalid or when the data is shared.

Index

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and Microsoft Word software. Proof pages were created on an Apple LaserWriter IINTX printer. Final pages negatives were output directly from text files on an Agfa ProSet 9800 imagesetter. Line art was created using Adobe Illustrator. PostScript®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

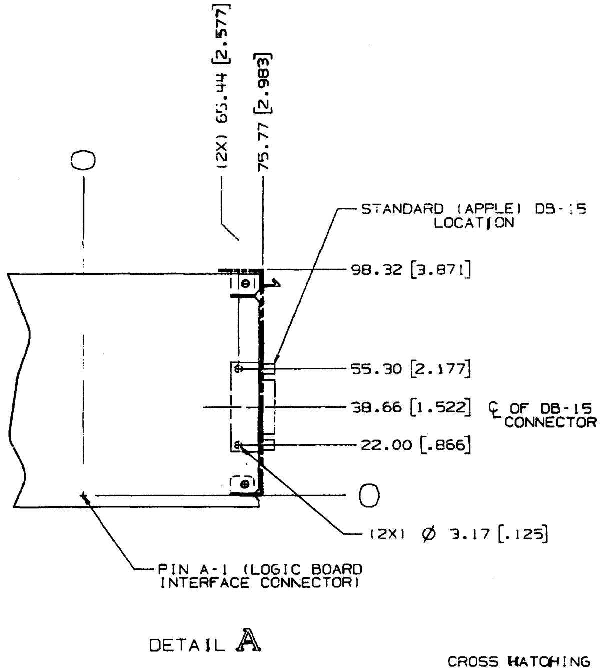
Text type and display type are Apple's corporate font, a condensed version of ITC Garamond®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

Writer: Cathy Donovan
Developmental Editor: Jeanne Woodward
Illustrator: Sandee Karr
Production: Janet M. Anders

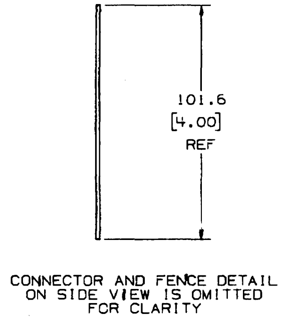
Special thanks to Jano Banks, Mark Baumwell, Rich Collyer, Jon Fitch, David Fung, Denis Hescox, Ron Hochsprung, Brian Howard, Jerry Katzung, Jon Krakower, Ann Nunziata, Noah Price, Craig Prouse, Mike Puckett, and Jim Stockdale for their technical assistance.

Also, a very special thanks to Rolly Reed and Roy Smith, writers of the first and second editions of this publication.

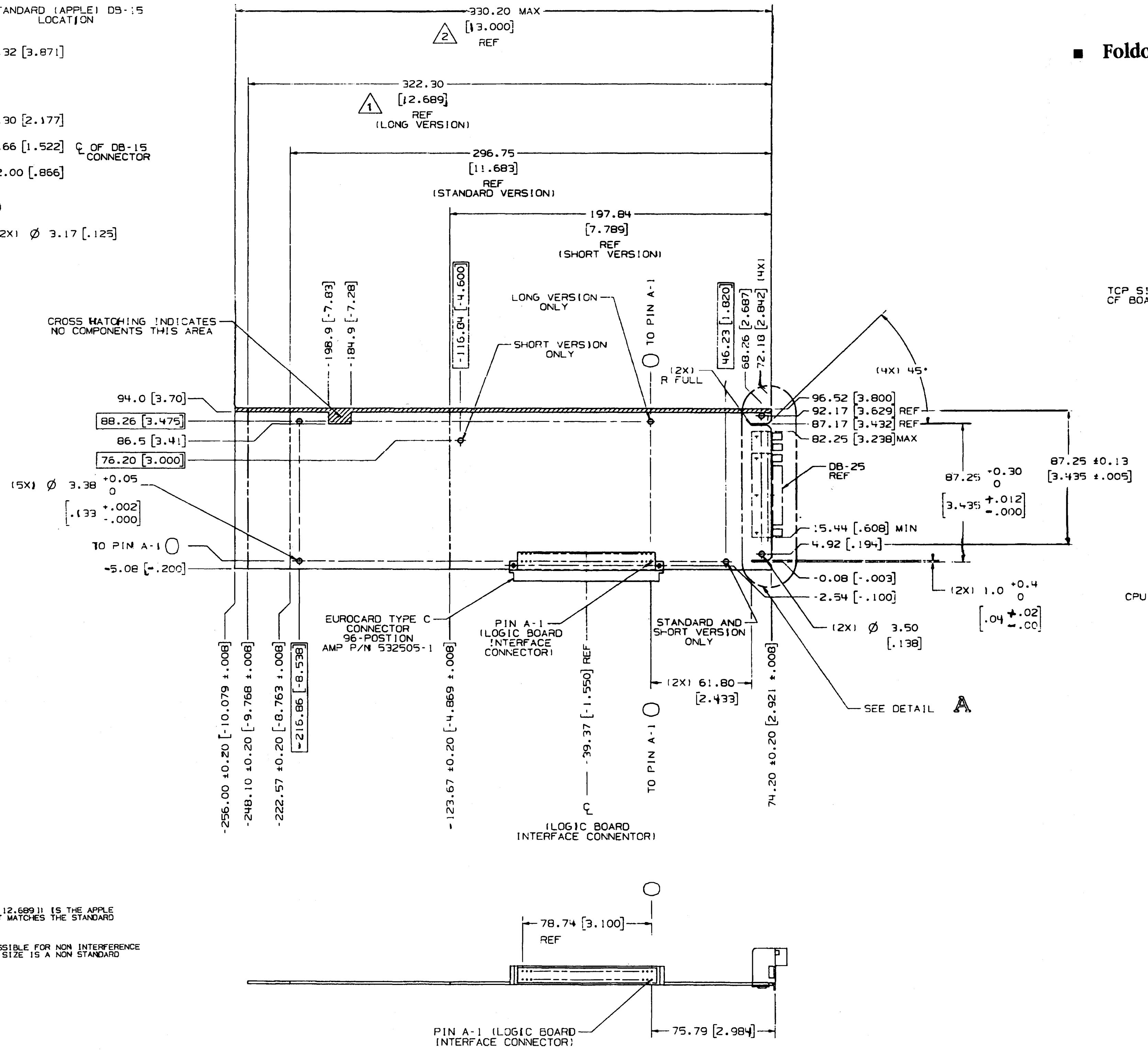
■ **Foldout 1** Design guide for Macintosh family standard NuBus cards

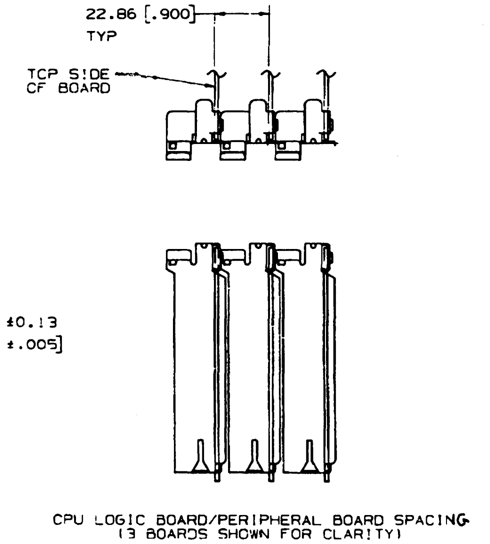


(2X) 65.44 [2.577]
75.77 [2.983]
STANDARD (APPLE) DB-15 LOCATION
98.32 [3.871]
55.30 [2.177]
38.66 [1.522] ℄ OF DB-15 CONNECTOR
22.00 [.866]
(2X) Ø 3.17 [.125]
PIN A-1 (LOGIC BOARD INTERFACE CONNECTOR)
DETAIL A



101.6
[4.00]
REF
CONNECTOR AND FENCE DETAIL ON SIDE VIEW IS OMITTED FOR CLARITY





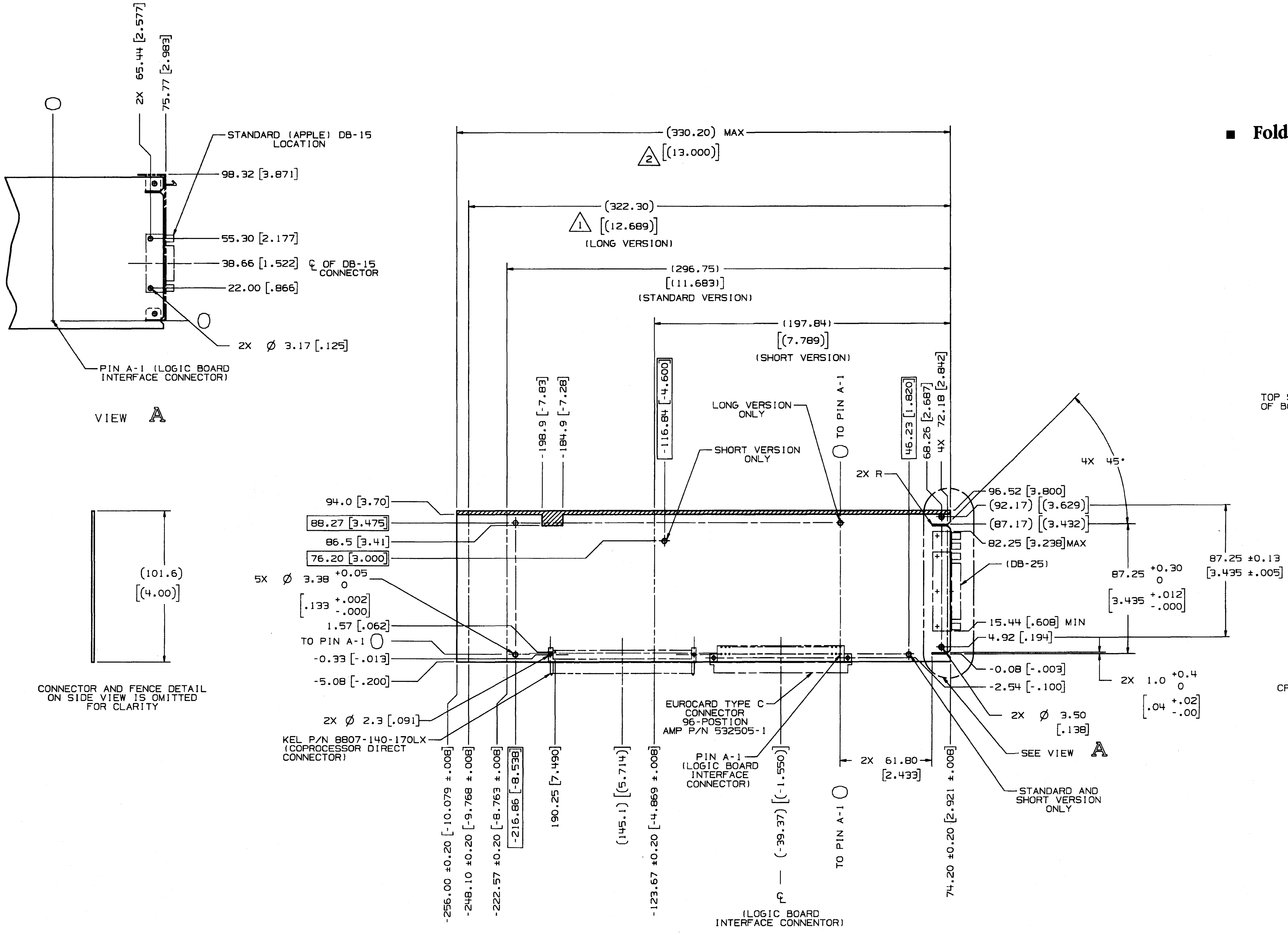
CPU LOGIC BOARD/PERIPHERAL BOARD SPACING (3 BOARDS SHOWN FOR CLARITY)

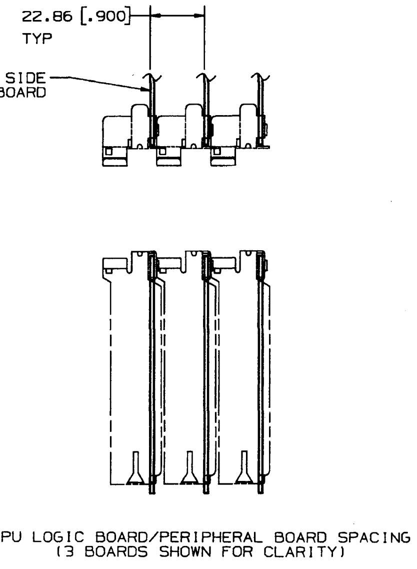
NOTE: UNLESS OTHERWISE SPECIFIED

1. THE LONG VERSION DIMENSION (322.30 [12.689]) IS THE APPLE PREFERRED MAXIMAUM BOARD LENGTH THAT MATCHES THE STANDARD PANEL SIZE.

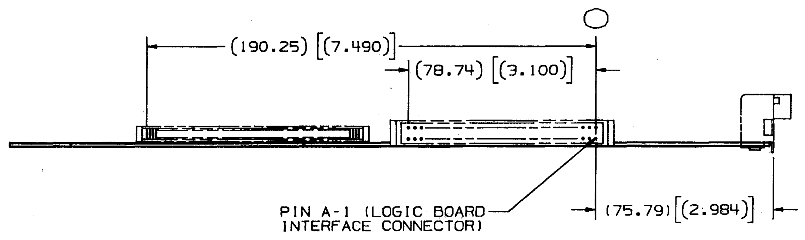
2. THIS DIMENSION MAXIMUM LENGTH PERMISSIBLE FOR NON INTERFERENCE WITH CPU INTERNAL COMPONENTS. THIS SIZE IS A NON STANDARD PANEL SIZE.

■ **Foldout 2** Design guide for Macintosh Quadra-family NuBus and PDS expansion cards





CPU LOGIC BOARD/PERIPHERAL BOARD SPACING (3 BOARDS SHOWN FOR CLARITY)



NOTES: UNLESS OTHERWISE SPECIFIED

1. THE LONG VERSION DIMENSION (322.30 [12.689]) IS THE APPLE PREFERRED MAXIMUM BOARD LENGTH THAT MATCHES THE STANDARD PANEL SIZE.

2. THIS DIMENSION IS THE MAX LENGTH PERMISSIBLE FOR NO INTERFERENCE WITH CPU INTERNAL COMPONENTS. THIS SIZE IS A NON-STANDARD PANEL SIZE.

■ **Foldout 3** NuBus card clearance requirements for Macintosh II, Macintosh IIx, Macintosh IIfx, and Macintosh Quadra 900 computers

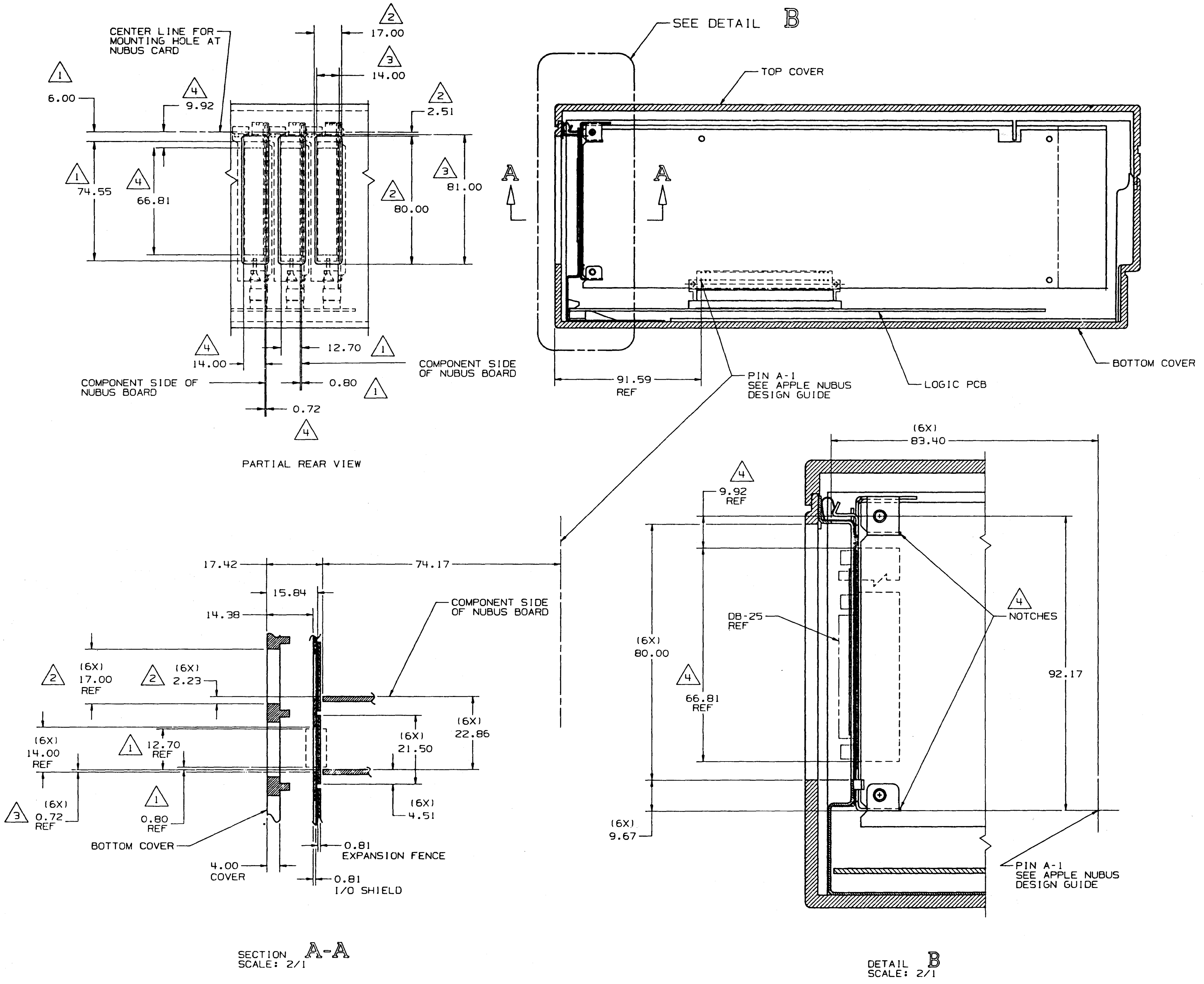
NOTE: UNLESS OTHERWISE SPECIFIED

△1 NUBUS BOARDS WHICH CONFORM TO THE ANSI/IEEE STD 1196 SPECIFICATION. (AS SHOWN IN THE PARTIAL REAR VIEW) FOR "MAXIMUM CONNECTOR CUTOUT" WILL PROPERLY FIT AS LONG AS THE DESIGN ALSO ALLOWS CLEARANCE FOR SURROUNDING PLASTIC AND I/O SHIELD. REFERENCE SECTION A-A AND DETAIL B.

△2 PLASTIC HOUSING CLEARANCE DIMENSIONS. THESE SHOULD BE USED AS DESIGN LIMITS FOR MATING CABLES OR COMPONENTS THAT REQUIRE REAR ACCESS.

△3 I/O SHIELD CLEARANCE DIMENSIONS. THESE SHOULD BE USED AS DESIGN LIMITS FOR BOARD MOUNTED CONNECTOR OR COMPONENTS THAT REQUIRE REAR ACCESS.

△4 APPLE COMPUTER NUBUS BOARD CONNECTOR CLEARANCE DIMENSIONS. THESE SHOULD BE USED AS DESIGN LIMITS FOR APPLE SPECIFICATION 062-0484. NUBUS BOARDS WITH NOTCHES, REFERENCE DETAIL B.



PARTIAL REAR VIEW

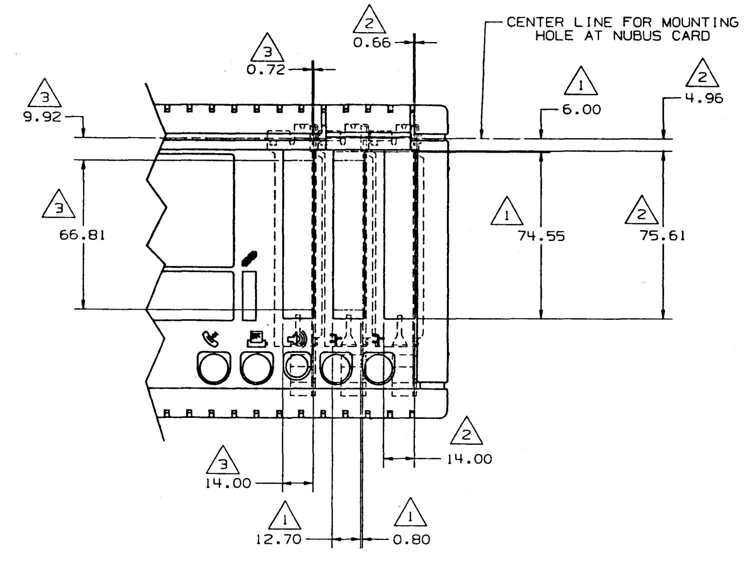
SECTION A-A
SCALE: 2/1

DETAIL B
SCALE: 2/1

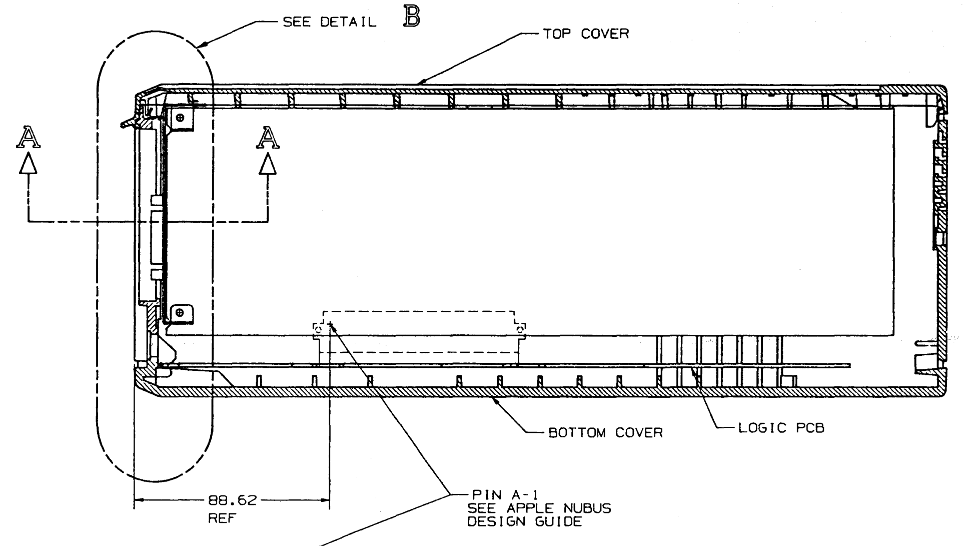
NOTE: UNLESS OTHERWISE SPECIFIED

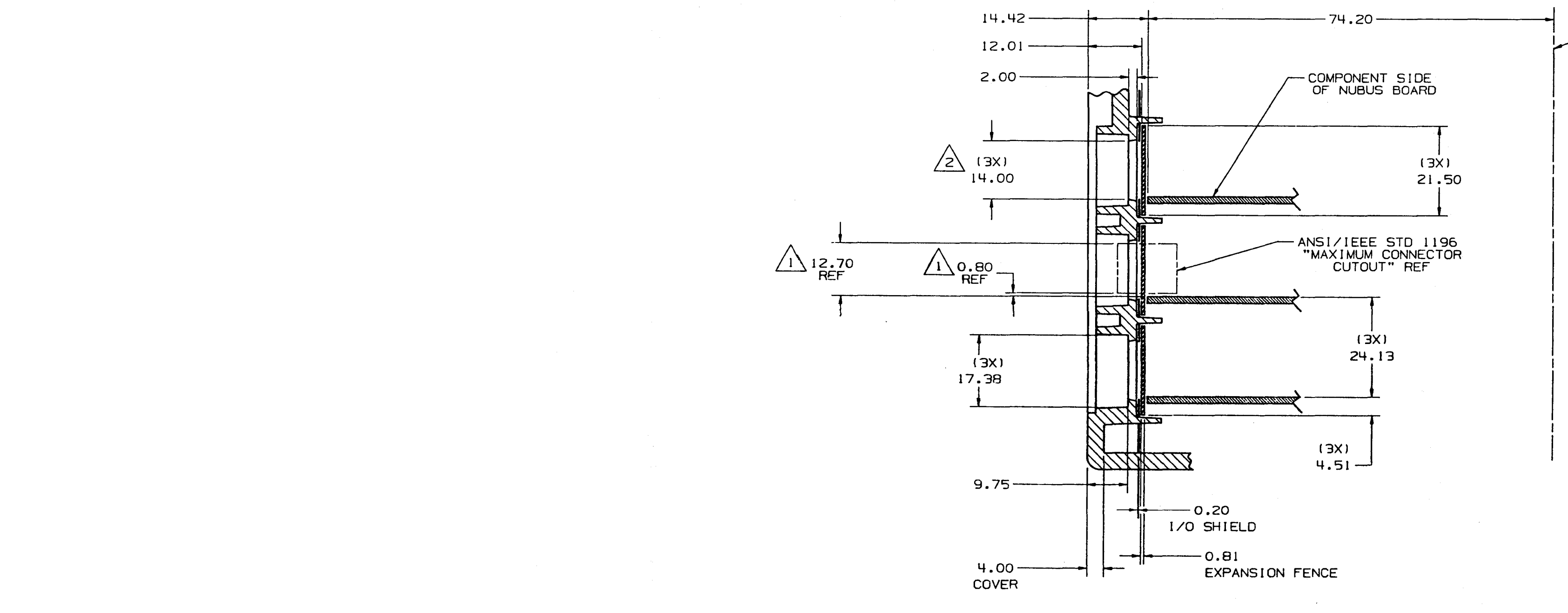
1. NUBUS BOARDS WHICH CONFORM TO THE ANSI/IEEE STD 1196 SPECIFICATION (AS SHOWN IN THE PARTIAL REAR), FOR "MAXIMUM CONNECTOR CUTOUT" WILL FIT PROPERLY AS LONG AS THE DESIGN ALSO ALLOWS CLEARANCE FOR SURROUNDING PLASTIC. REFERENCE SECTION A-A AND DETAIL B.
2. PLASTIC HOUSING CLEARANCE DIMENSIONS FOR THE MAC IIcx. THESE SHOULD BE USED AS DESIGN LIMITS FOR BOARD MOUNTED CONNECTORS OR COMPONENTS AND MATING CABLES.
3. APPLE COMPUTER NUBUS BOARD CONNECTOR CLEARANCE DIMENSIONS THESE SHOULD BE USED AS DESIGN LIMITS FOR APPLE SPECIFICATION 062-0484. NUBUS BOARDS WITH NOTCHES, REFERENCE DETAIL B.

■ **Foldout 4** NuBus card clearance requirements for Macintosh IIcx, Macintosh IIci, and Macintosh Quadra 700 computers

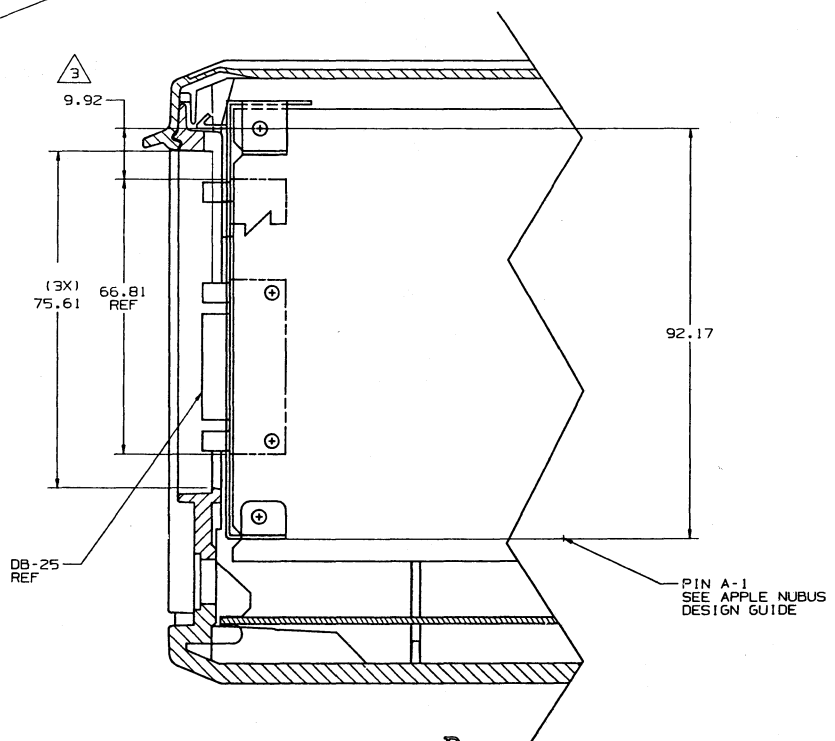


PARTIAL REAR VIEW





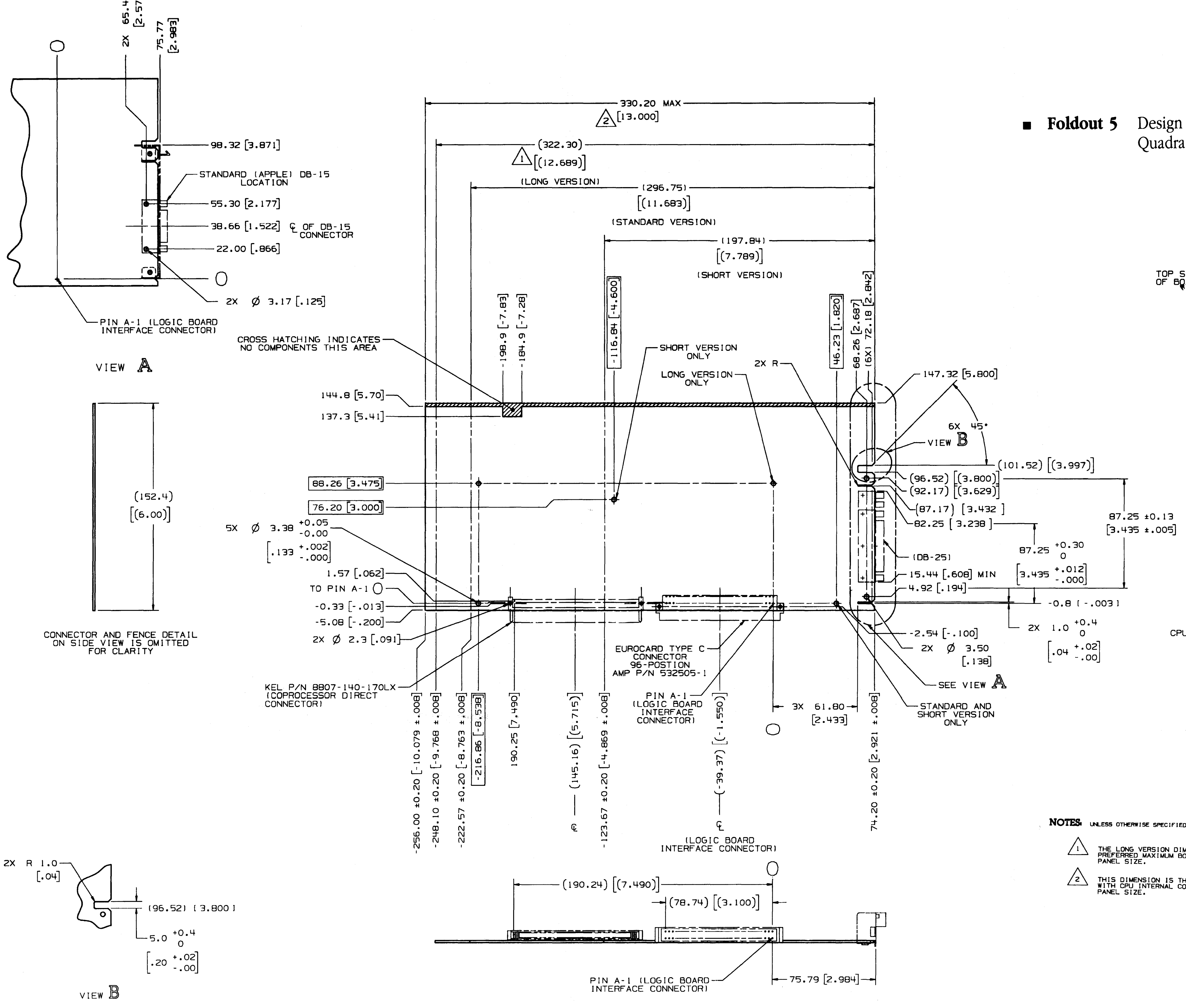
SECTION A-A
SCALE: 2/1

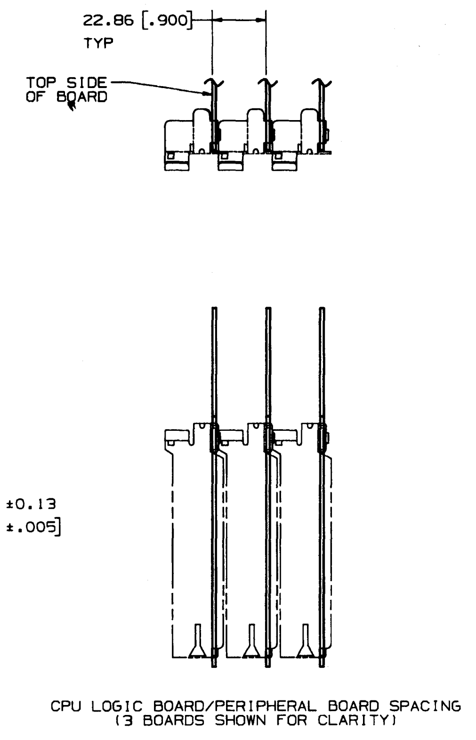


DETAIL B
SCALE: 2/1

CAD GENERATED

■ **Foldout 5** Design guide for oversized Macintosh Quadra 900 NuBus card





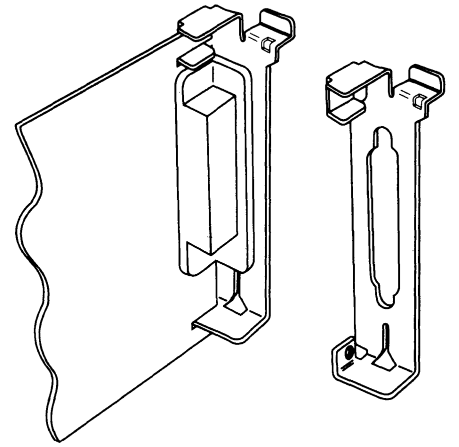
CPU LOGIC BOARD/PERIPHERAL BOARD SPACING (3 BOARDS SHOWN FOR CLARITY)

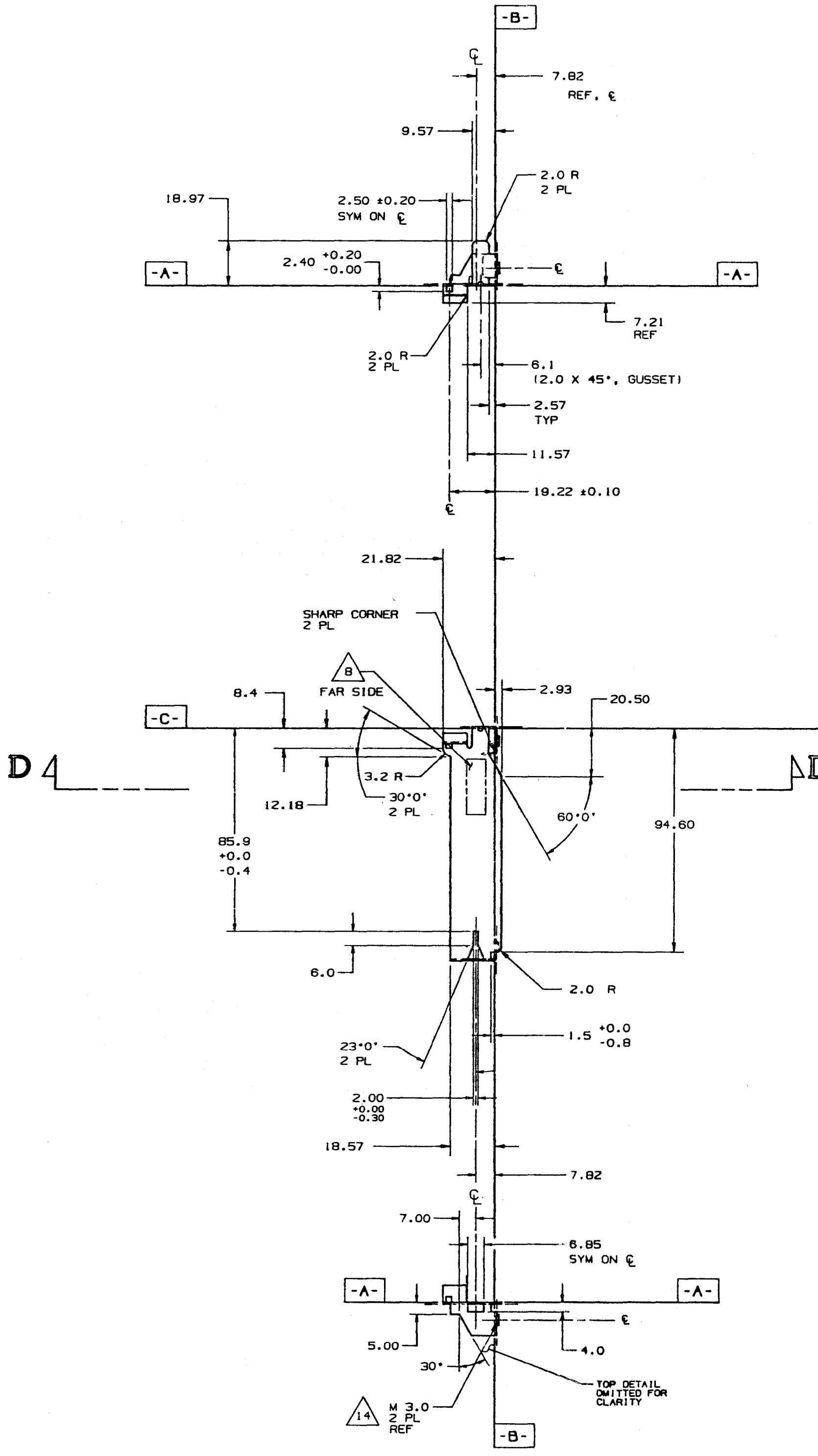
NOTES: UNLESS OTHERWISE SPECIFIED

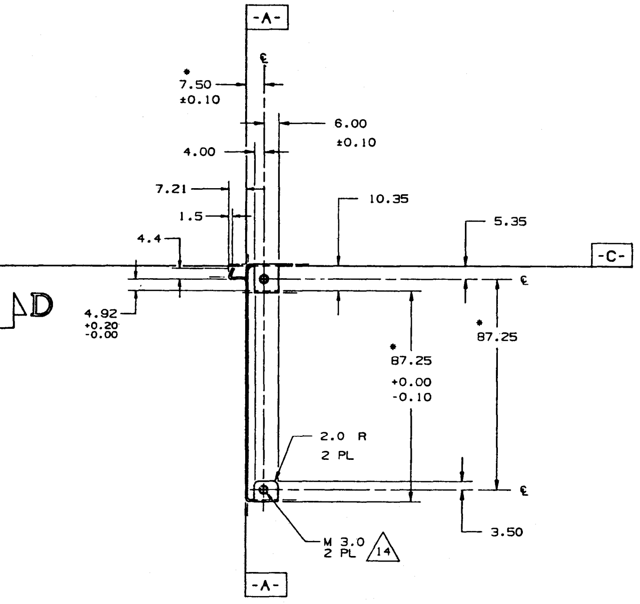
1. THE LONG VERSION DIMENSION (322.30 [12.689]) IS THE APPLE PREFERRED MAXIMUM BOARD LENGTH THAT MATCHES THE STANDARD PANEL SIZE.

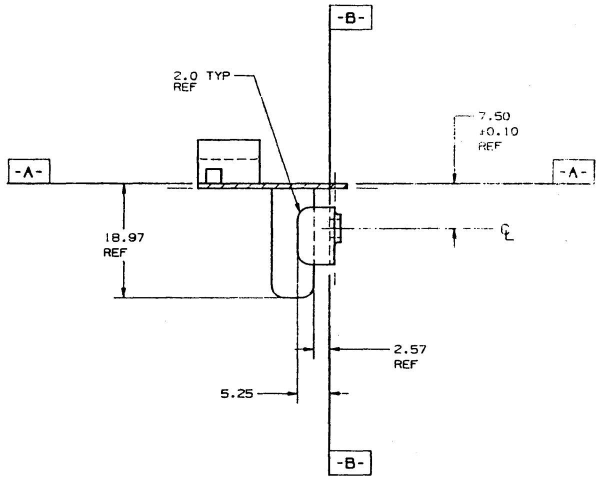
2. THIS DIMENSION IS THE MAX LENGTH PERMISSIBLE FOR NO INTERFERENCE WITH CPU INTERNAL COMPONENTS. THIS SIZE IS A NON-STANDARD PANEL SIZE.

- **Foldout 6** Connector shield for Macintosh II-family computer





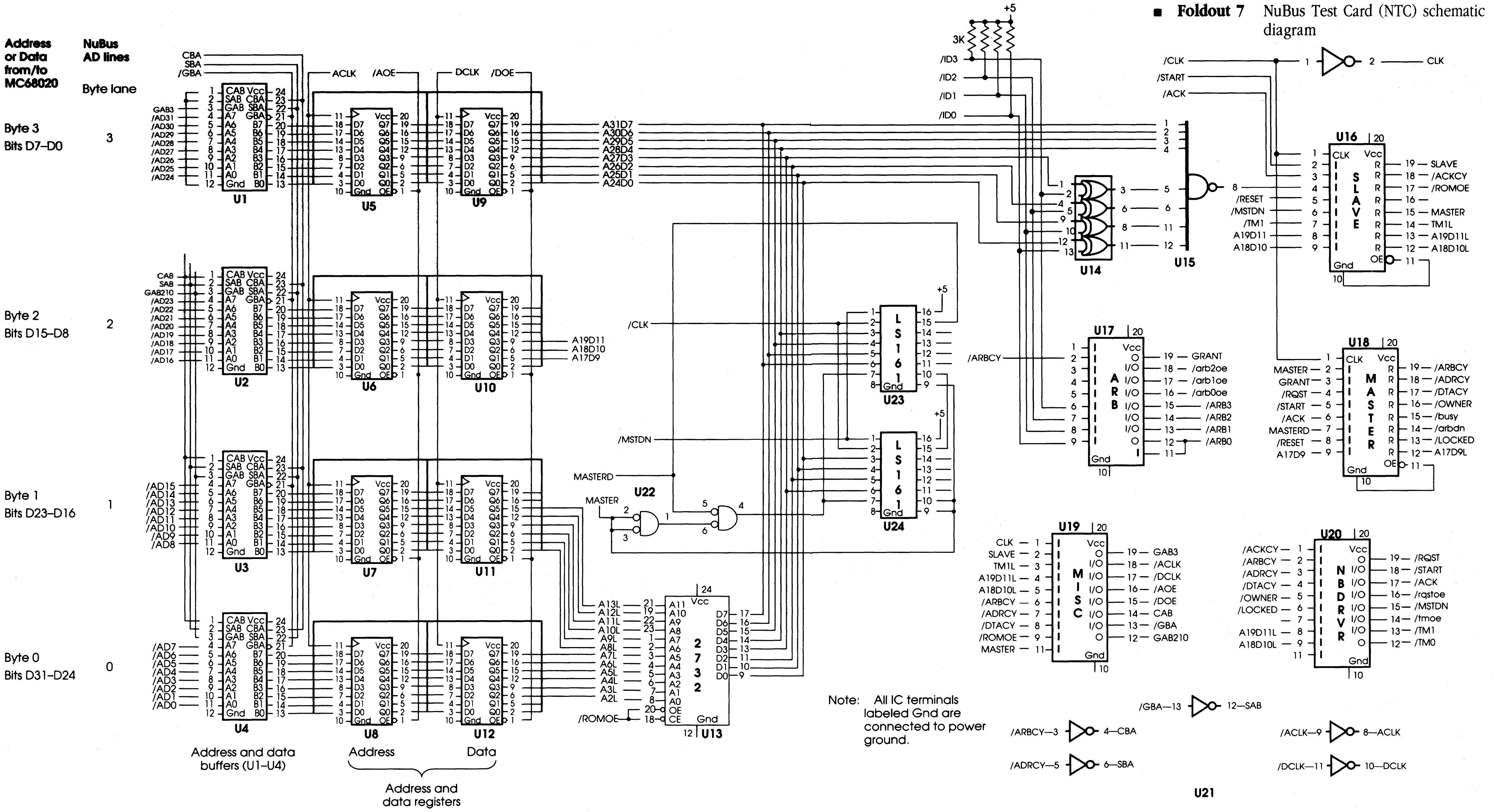




SECTION **D-D**

SCALE: 2.5

■ **Foldout 7** NuBus Test Card (NTC) schematic diagram



Note: All IC terminals labeled Gnd are connected to power ground.

■ **Foldout 8** Connector card design guide for Macintosh PDS computers

NOTE: UNLESS OTHERWISE SPECIFIED

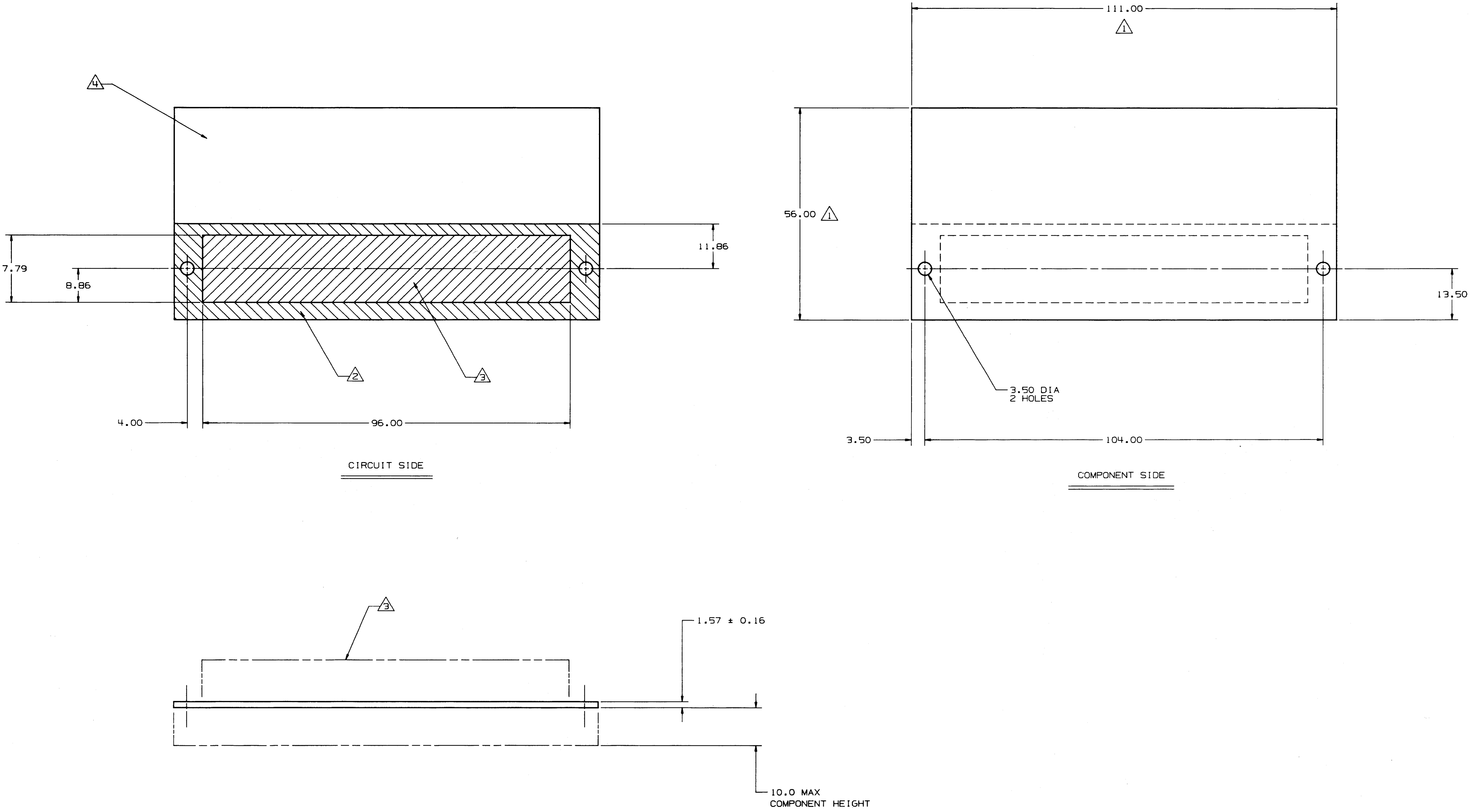
△1 INDICATED DIMENSIONS SPECIFY MAXIMUM BOARD OUTLINE. NO COMPONENTS, INCLUDING MATING CONNECTOR FROM EXPANSION BOARD, TO PROTRUDE BEYOND THE BOARD EDGES.

△2 INDICATED AREA WILL CONTACT THE METAL CHASSIS WHEN BOARD IS INSTALLED. NO TRACES OR FEEDTHROUGHS ARE PERMITTED.

△3 INDICATED AREA IS RESERVED FOR I/O CONNECTORS, WHICH ARE TO BE MOUNTED ON THE CIRCUIT SIDE AS SHOWN. NO OTHER COMPONENTS PERMITTED ON CIRCUIT SIDE.

△4 MAXIMUM LEAD LENGTH AFTER SOLDERING TO BE 2.5 MM.

5. WARP AND TWIST OF FINISHED BOARD NOT OT EXCEED 0.25 MM (.010 IN.) PER INCH WHEN MEASURED IN ACCORDANCE WITH IPC-A-600.



CIRCUIT SIDE

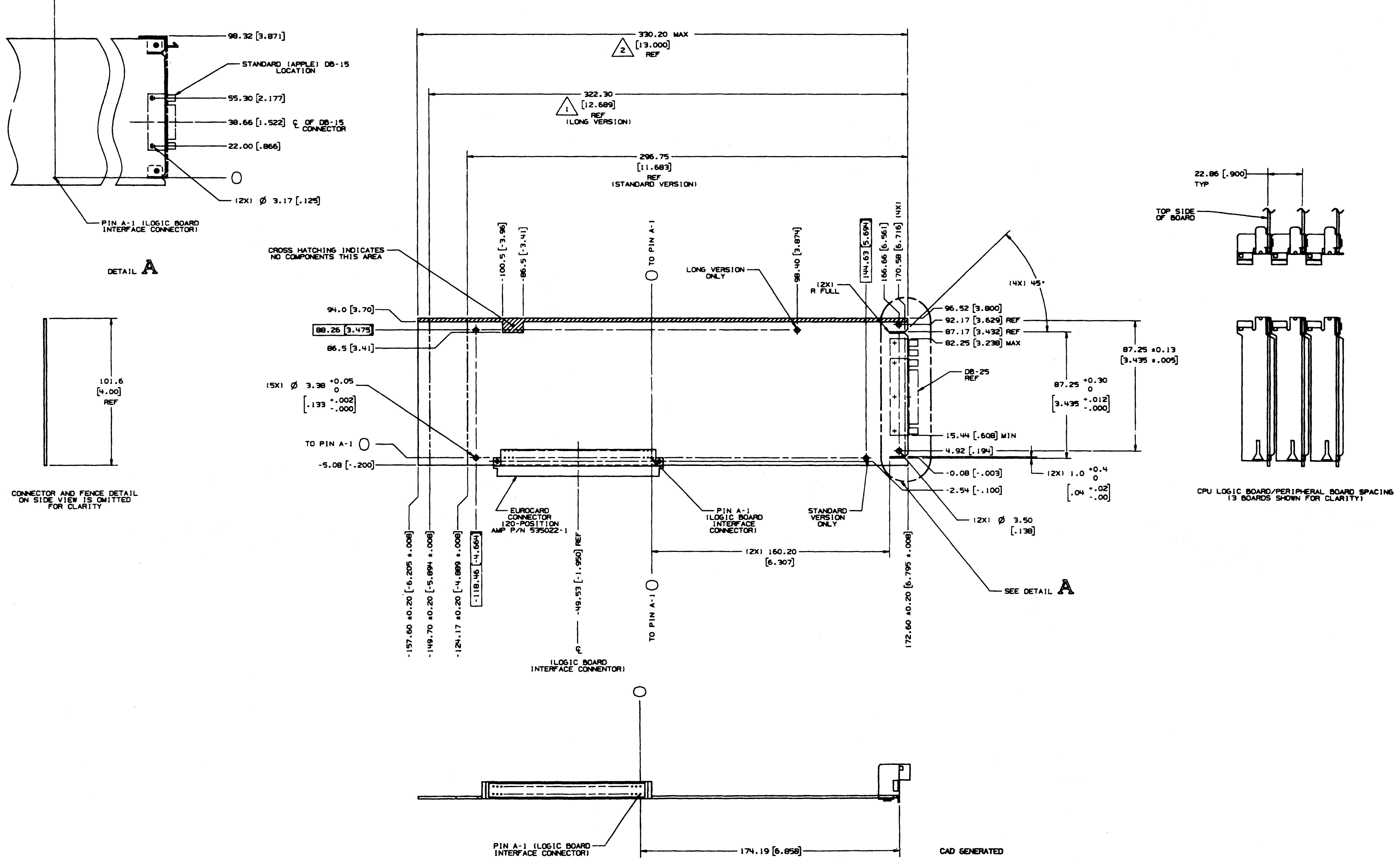
COMPONENT SIDE

NOTE: UNLESS OTHERWISE SPECIFIED

1. THE LONG VERSION DIMENSION (322.30 [12.609]) IS THE APPLE PREFERRED MAXIMAUM BOARD LENGTH THAT MATCHES THE STANDARD PANEL SIZE.

2. THIS DIMENSION MAXIMUM LENGTH PERMISSIBLE FOR NON INTERFERENCE WITH CPU INTERNAL COMPONENTS. THIS SIZE IS A NON STANDARD PANEL SIZE.

■ **Foldout 9** Design guide for Macintosh IIfx PDS expansion card



> $29.95 USA
> $38.95 CANADA

The Official Publication from Apple Computer, Inc.

Designing Cards and Drivers for the Macintosh Family
Third Edition

In continuing Apple's tradition of providing new product information, *Designing Cards and Drivers for the Macintosh Family,* third edition, gives you up-to-date expansion guidelines for all expandable Macintosh models, including the six new models introduced in late 1991. A companion book, *Guide to the Macintosh Family Hardware*, provides detailed information on the hardware design of the Macintosh family of computers.

The third edition includes comprehensive guidelines for designing expansion cards and drivers for the following Macintosh models:

- Macintosh SE
- Macintosh SE/30
- Macintosh II
- Macintosh IIx
- Macintosh IIcx
- Macintosh IIci
- Macintosh IIsi
- Macintosh IIfx
- Macintosh LC
- Macintosh Classic
- Macintosh Classic II
- Macintosh Portable
- Macintosh PowerBook 100
- Macintosh PowerBook 140
- Macintosh PowerBook 170
- Macintosh Quadra 700
- Macintosh Quadra 900

This book consists of an introduction providing an overview of Apple's expansion strategy, three main parts, and five appendixes.

Part I describes the implementation of the NuBus™ interface, provides electrical and mechanical guidelines for designing NuBus expansion cards, and supplies information that is essential to the design of declaration ROM and driver software.

Part II describes the processor-direct slot (PDS) expansion interface and provides guidelines for designing expansion cards for the 68000, 68020, 68030, and 68040 Direct Slot expansion interfaces used in current Macintosh computers.

Part III discusses application-specific expansion interfaces and describes how they are implemented in the Macintosh family of computers to satisfy a unique purpose.

The appendixes provide information on a variety of subjects such as electromagnetic interference, heat dissipation, and safety standards; sample code for a declaration ROM and video drivers; and PAL listings for NuBus and SCSI-NuBus test cards. The book also contains a glossary of technical terms and an index.

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
408-996-1010
TLX 171-576

Addison-Wesley Publishing Company

52995
9 780201 608557

ISBN 0-201-60855-3
60855