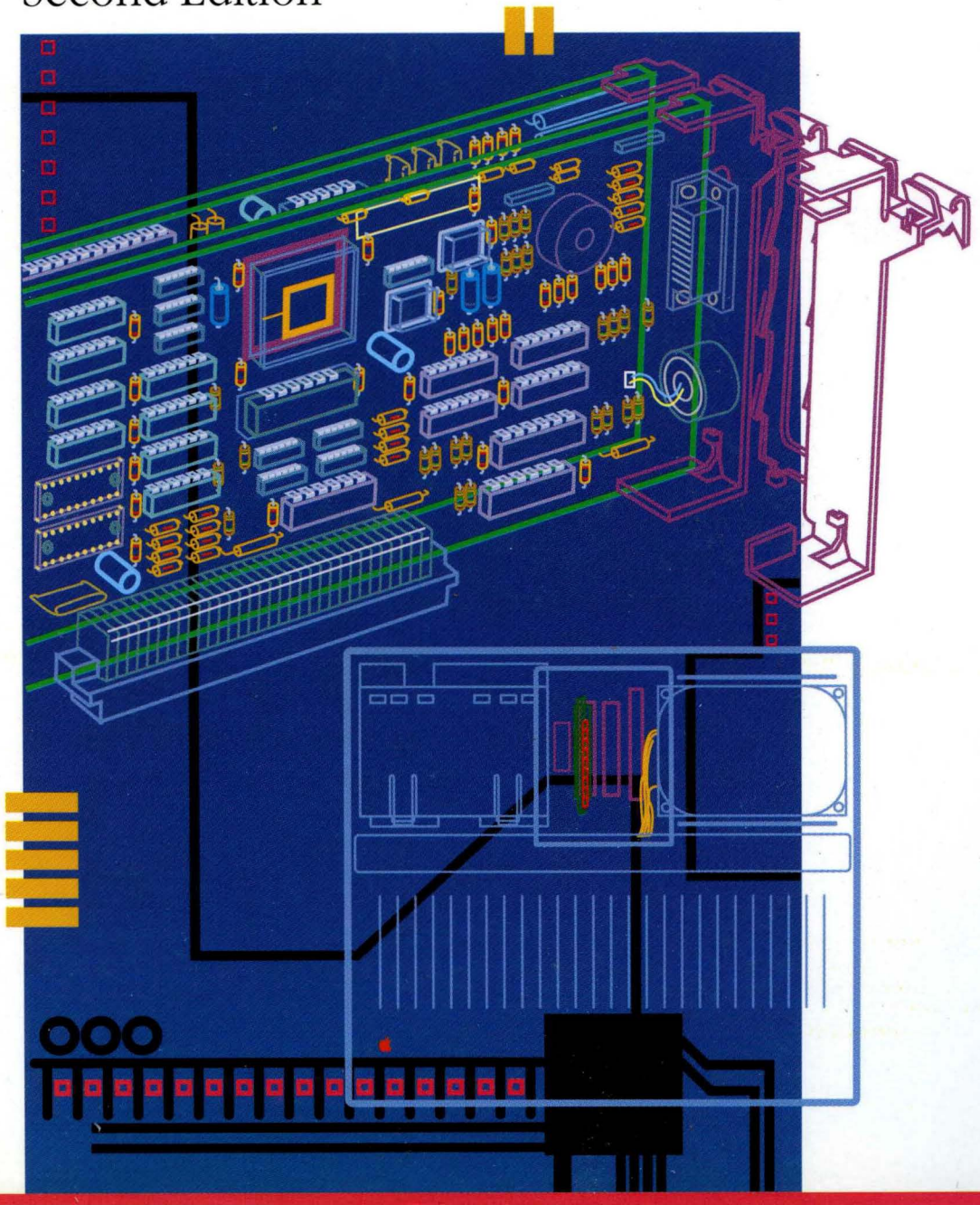
For Macintosh SE, Macintosh SE/30, Macintosh II, Macintosh IIx, Macintosh Portable, Macintosh IIcx, Macintosh IIci, and Macintosh IIfx

Designing Cards and Drivers for the Macintosh® Family

by Apple Computer, Inc.

Second Edition



Designing Cards and Drivers for the Macintosh® Family

Second Edition

For Macintosh SE, Macintosh SE/30, Macintosh II, Macintosh IIx, Macintosh Portable, Macintosh IIcx, Macintosh IIci, and Macintosh IIfx

Addison-Wesley Publishing Company, Inc.

Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn
Sydney Singapore Tokyo Madrid San Juan

20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Simultaneously published in the United States and Canada.

ISBN 0-201-52404-X
ABCDEFGHIJ-MU-9543210
First printing, May 1990

WARRANTY INFORMATION

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

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 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** Warnings like this indicate potential problems. ▲

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.

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 and MC68030 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,* and *68030 Direct Slot* are all used to identify the processor-direct expansion interface associated with some Macintosh computers. Other documents may use the terms *processor dependent slot* and *030 Direct Slot* to identify this interface.

The following abbreviations are used:

KB	kilobyte
Kbit	kilobit
MB	megabyte
Mbit	megabit
GB	gigabyte
ms	millisecond
µs	microsecond
ns	nanosecond
kΩ	kilohm
mA	milliamp
µA	microamp
pF	picofarad
K	1024

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.

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 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.

As the variety and the sophistication of Macintosh computers evolve, so does the documentation. In order 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 Macintosh Family Hardware,* second edition, and this book are two of the books in this suite.

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)
Introductory books	
Technical Introduction to the Macintosh Family	Introduction to the Macintosh software and hardware for the original Macintosh, the Macintosh Plus, the Macintosh SE, and the Macintosh II
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, second edition	Hardware and device-driver reference for the expansion capabilities of the Macintosh computer family including the Macintosh II, Macintosh IIx, Macintosh IIcx, Macintosh SE, Macintosh SE/30, Macintosh Portable, and Macintosh IIfx
Guide to Macintosh Family Hardware, second edition	Hardware reference and developer's guide for the Macintosh computer family including the Macintosh II, Macintosh IIx, Macintosh IIcx, Macintosh SE, Macintosh SE/30, Macintosh Portable, and Macintosh IIfx
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	Description of the Macintosh Programmer's Workshop (MPW™), Apple's software development environment for all Macintosh computers

You may also find useful information in the following earlier books, now superseded:

Macintosh Family Hardware Reference	Earlier reference for the Macintosh hardware for the classic Macintosh, Macintosh SE, and Macintosh II
Designing Cards and Drivers for Macintosh II and Macintosh SE	Earlier hardware and device driver reference for the expansion capabilities of the Macintosh II and Macintosh SE

How to get more information

Several organizations exist that provide support for Macintosh hardware and software developers. This section tells you how to contact the Apple Programmers and Developers Association (APDA™), Apple user groups, and Apple Developer Services.

APDA

APDA provides a wide range of technical products and documentation, from Apple and other suppliers, for programmers and developers who work on Apple equipment. For information about APDA, contact

APDA
Apple Computer, Inc.
20525 Mariani Avenue, Mailstop 33-G
Cupertino, CA 95014
800-282-APDA (800-282-2732)
FAX: 408-562-3971
Telex: 171-576
AppleLink: 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 Machintosh Technical Notes, pertinent Developer Program information, and all the latest news relating to Apple products. You will also receive Apple's Technical Guide Book 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, Mailstop 51-W
Cupertino, CA 95014

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 II family of modular computers and can be supported across a variety of Macintosh products.

Macintosh compact computers such as the Macintosh SE and the Macintosh SE/30 use the processor-direct slot (PDS) expansion interface. The Macintosh IIfx has a processor-direct slot, but its primary expansion interface is the NuBus. The Macintosh Portable also has a processor-direct slot, but its usefulness is somewhat limited in comparison to 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 will probably change 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.

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. All Macintosh II–family computers include the NuBus expansion interface in configurations ranging from three 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.

You can expect enhancements to Apple's implementation of NuBus on future Macintosh machines. These could include block transfer modes and other new features that are not available on current Macintosh computers with NuBus. 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** expansion interface on compact, or small-footprint, Macintosh computers such as the Macintosh SE and the Macintosh SE/30, or any design for which NuBus is inappropriate. The Macintosh IIfx also includes a processor-direct slot but its 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 plans to limit the number of PDS configurations that you must support by allowing only one slot specification for each of the microprocessors used. The goal is to have the PDS specification remain constant within a microprocessor family, and 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 Direct Slot expansion interface

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 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 more powerful MC68030 microprocessor used in these machines. 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 the 68000 Direct Slot will not work in the 68030 Direct Slot, and vice versa.

Recommended strategy for 68030 Direct Slot expansion card design

The MC68030-based Macintosh SE/30 and Macintosh IIfx are currently the only Macintosh computers with the 68030 PDS hardware expansion capability, but plans call for future MC68030-based Macintosh machines with and without NuBus to also implement the 68030 Direct Slot.

The 68030 Direct Slot electrical specification contains 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 Direct Slot. Others, however, are classified as *machine specific,* meaning that they may or may not be present on different Macintosh computers that use the 68030 Direct Slot. 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 14, "Electrical Design Guide for 68030 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 II–family computers. This method of emulating NuBus address space is called *pseudo-slot design.* If you follow the pseudo-slot 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 Direct Slot 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 14 provides more detailed information on pseudo-slot expansion card design.

If you don't take advantage of pseudo-slot design, you have to do several things differently. Apple has reserved a range of physical address spaces in the memory map for 68030 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 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 cards that take advantage of the pseudo-slot design features will not work when used in a NuBus slot of a Macintosh with the NuBus interface because of bus conflicts with physical NuBus. However, pseudo-slot design allows you to easily convert your 68030 Direct Slot expansion card to a NuBus card, or vice versa. (The similarity is in the card's electrical and software designs; NuBus and PDS cards have different card sizes and connectors, so the mechanical design would have to be changed to allow the redesigned card to fit the target slot.)

Application-specific expansion

The Macintosh computers that offer application-specific expansion usually have 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 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 Macintosh 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 Direct Slot via the pseudo-slot method, you need to supply only one driver for both 68030 Direct Slot cards and NuBus cards, and 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; that implementation is an extension of the Texas Instruments product documented in their NuBus Specification, document number TI-2242825-0001*A, copyright 1983. Some features of NuBus, most notably block data transfer and system parity valid, are documented for completeness even though they are not supported in the Macintosh II family of computers; these instances 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 computer in the Macintosh II family, a comparison of major features, and 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 II–family computer.

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. Pseudo-code 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 three 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 the PAL listings for the NuBus Test Card described in Chapter 10.

Appendix C 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 the Macintosh II–family computers. Certain features of the NuBus interface are not implemented in the Macintosh II–family 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 will probably also need the *IEEE Standard for a Simple 32-Bit Backplane Bus: NuBus, ANSI/IEEE Std 1196-1987.* It may be ordered from

Institute of Electrical and Electronics Engineers
345 East 47 Street
New York, NY 10017

Texas Instruments owns patents on the NuBus. If you wish to make devices for computers with the NuBus interface, 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 Hardware Overview of the Macintosh II Family

This chapter provides a basic description of the structure and organization of the Apple Macintosh II family of computers. The Macintosh II, the Macintosh IIx, the Macintosh IIcx, the Macintosh IIci, and the Macintosh IIfx are in this computer family. All use an I/O bus based on 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. The chapter places the NuBus interface in context within the total computing machine.

Major features

Table 1-1 compares the major features of the Macintosh II–family computers.

■ **Table 1-1** Major features of the Macintosh II family

Feature	Macintosh II	Macintosh IIx	Macintosh IIcx	Macintosh IIci	Macintosh IIfx
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	MC68030 32-bit address bus 32-bit data bus
Processor clock	15.6672 MHz	15.6672 MHz	15.6672 MHz	25 MHz	40 MHz
Coprocessor	MC68881 floating-point unit	MC68882 floating-point unit	MC68882 floating-point unit	MC68882 floating-point unit	MC68882 floating-point unit
Memory management	24-to-32-bit address translation by AMU; or logical-to-physical address translation by optional MC68851 PMMU	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	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	On-board video with main memory screen buffer or NuBus video card	NuBus video card with on-card screen buffer
RAM	Up to 128 MB of DRAM in eight 30-pin SIMMs (over 2 GB available through NuBus slots)	Up to 128 MB of DRAM in eight 30-pin SIMMs (over 2 GB available through NuBus slots)	Up to 128 MB of DRAM in eight 30-pin SIMMs (over 2 GB available through NuBus slots)	Up to 128 MB of DRAM in eight 30-pin SIMMs (over 2 GB available through NuBus slots); optional DRAM parity with 9-bit SIMMs	Up to 128 MB of DRAM in 64-pin SIMMs (over 2 GB available through NuBus slots); 32 KB of SRAM, fast RAM cache; optional DRAM parity with 9-bit SIMMs

(Continued)

■ **Table 1-1** Major features of the Macintosh II family (Continued)

Feature	Macintosh II	Macintosh IIx	Macintosh IIcx	Macintosh IIci	Macintosh IIfx
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	512 KB standard in one ROM SIMM
Expansion slots	Six NuBus slots	Six NuBus slots	Three NuBus slots	Three NuBus slots, one cache card connector	Six NuBus slots, one processor-direct slot
Keyboard and mouse interface	Two Apple Desktop Bus™ (ADB) ports	Two Apple Desktop Bus (ADB) ports	Two Apple Desktop Bus (ADB) ports	Two Apple Desktop Bus (ADB) ports	Two Apple Desktop Bus (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	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" FDHD™ drives (one standard, one optional)	Super Woz Integrated Machine (SWIM) chip controls one internal 1.4 MB, 3.5" drive and one optional external FDHD drive	Super Woz Integrated Machine (SWIM) chip controls one internal 1.4 MB, 3.5" drive and one optional external FDHD drive	Super Woz Integrated Machine (SWIM) chip controls one internal 1.4 MB, 3.5" drive and one optional external FDHD drive
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	one internal 50-pin, one external DB-25
Sound	Apple Sound Chip	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	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 Macintosh Family Hardware* manual (which supersedes the *Macintosh Family Hardware Reference*). The *Technical Introduction to the Macintosh Family* contains a higher level overview.

Block diagrams of the computers in the Macintosh II family are shown in Figures 1-1 through 1-4. 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 more powerful Motorola MC68030 microprocessor , which is also driven by a 15.6672 MHz clock. The Macintosh IIci (Figure 1-3) uses an MC68030 microprocessor but its clock speed has been increased to 25 MHz. The Macintosh IIfx (Figure 1-4), the most powerful member of the Macintosh II family, also uses an MC68030 microprocessor but it is driven by a 40 MHz clock.

All members of the Macintosh II family 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 1-4 and include

- a 65C23 Versatile Interface Adapter (VIA1) for communicating with the ADB transceiver which, in turn, communicates with the mouse and keyboard (the VIA also communicates with the real-time clock)
- another 65C23 Versatile Interface Adapter (VIA2) for handling interrupts from the NuBus slots (not used in the Macintosh IIci or Macintosh IIfx)
- an NCR 53C80 SCSI (Small Computer System Interface) chip for high-speed data transfer with the internal hard disk and any other SCSI device (integrated into SCSI DMA chip on the Macintosh IIfx)
- a Zilog Z8530 Serial Communications Controller (SCC) for serial communication
- 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, and Macintosh IIfx and controls the 1.4 MB, 3.5-inch FDHD (floppy disk, high density) drives as well as 800 KB floppy disk drives (an FDHD drive is also called a SuperDrive™)
- the Apple Sound Chip (ASC) sound generator

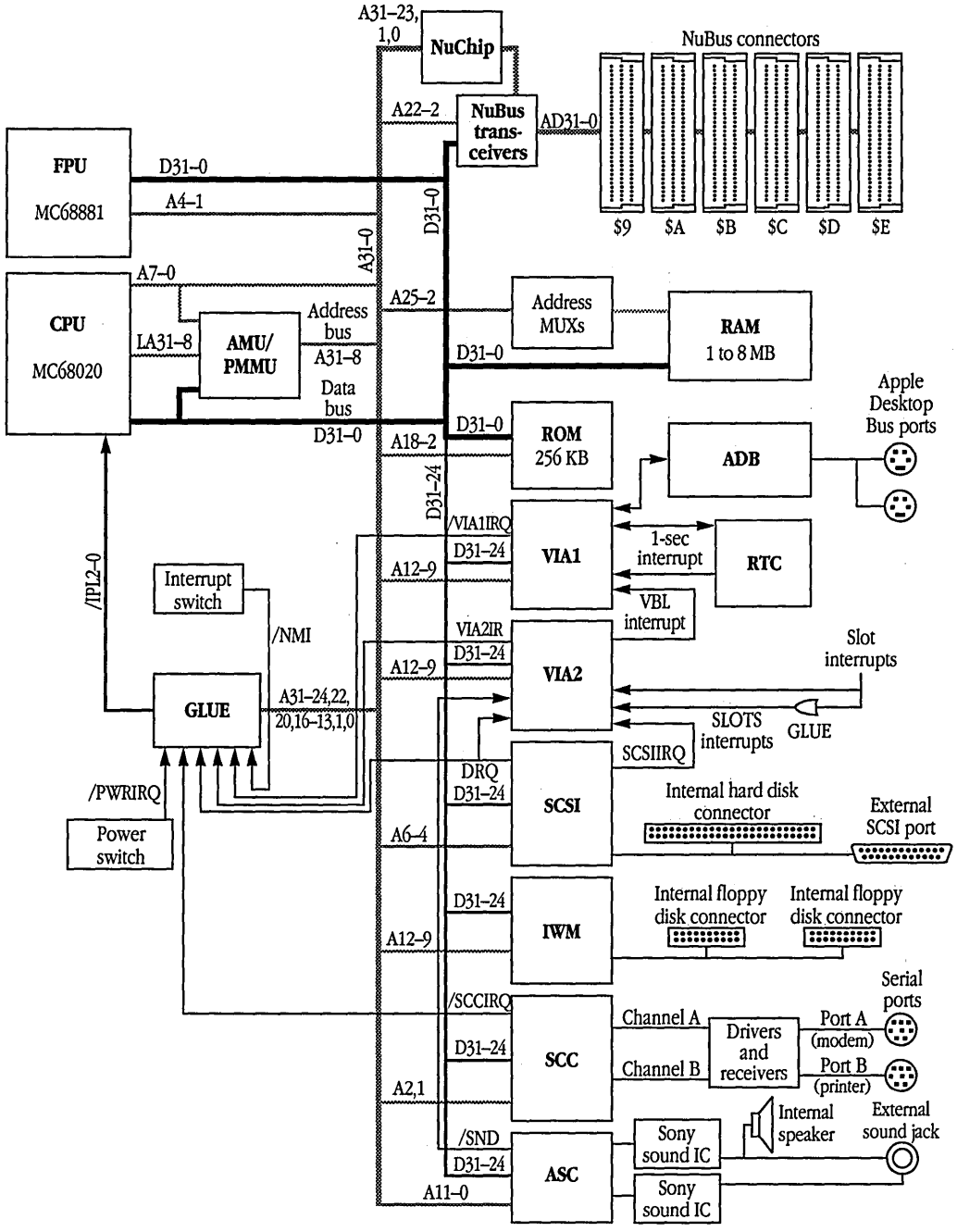
In addition, the Macintosh IIfx includes several new Apple custom ICs that enable the microprocessor to communicate with external devices. These ICs are shown in Figure 1-4 and include

- a SCSI DMA chip, which not only provides all of the functions of the NCR 53C80 SCSI chip used in the other Macintosh II–family 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 communication 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 services) chip that handles interrupts and device decodes, functions that are performed by the VIAs and the GLUE chip in the other Macintosh II–family computers

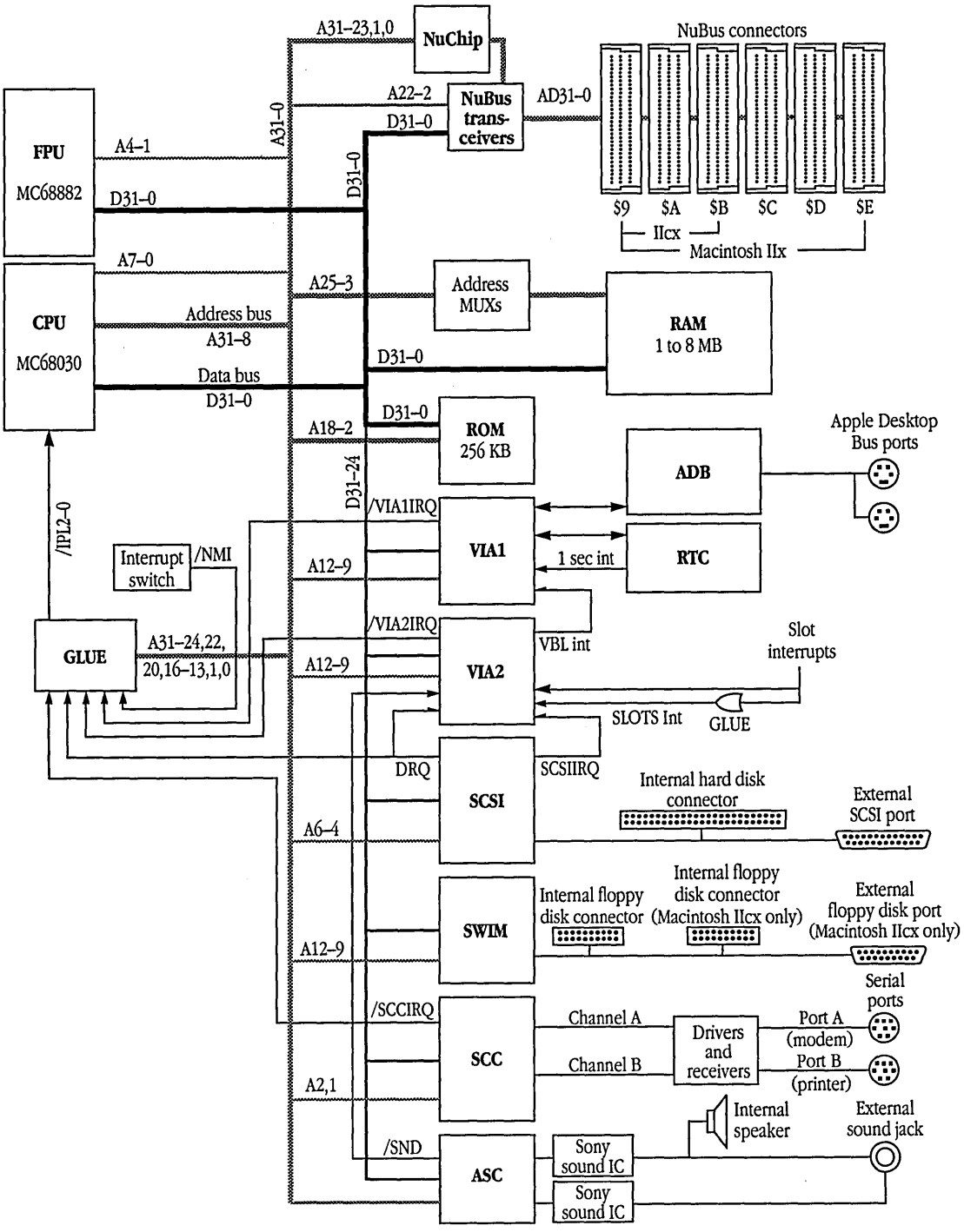
In all Macintosh II–family computers except the Macintosh IIci, the interface with the video display is through a video card in one of the NuBus slots. The Macintosh IIci has the option of using a NuBus video card, but its primary video interface is built into the computer. Video signals are generated 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 floating-point numeric coprocessors (MC68881 for the Macintosh II and MC68882 for the other computers in the Macintosh II family) use the coprocessor interfaces of their respective microprocessors.

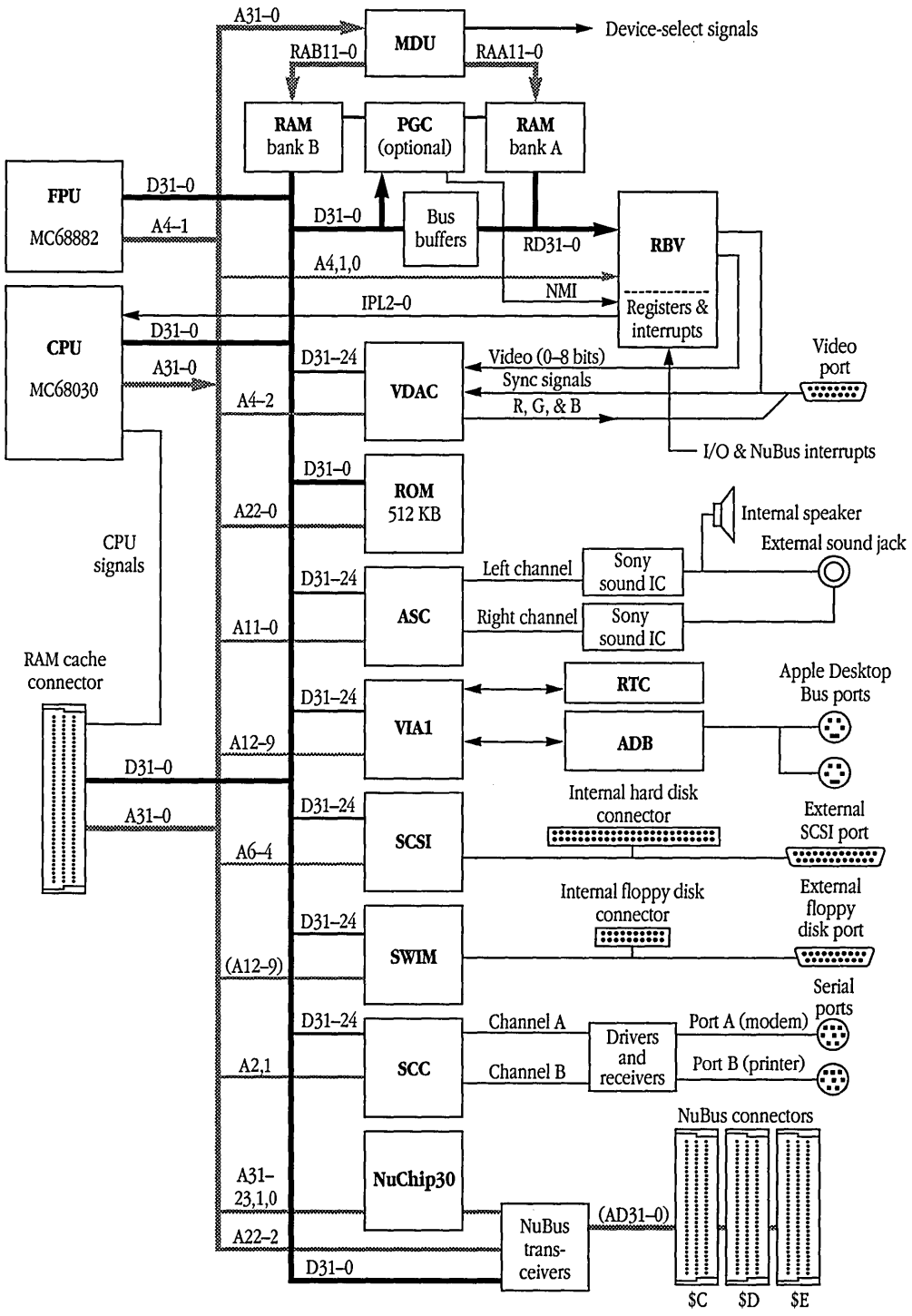
■ **Figure 1-1** Block diagram of the Macintosh II



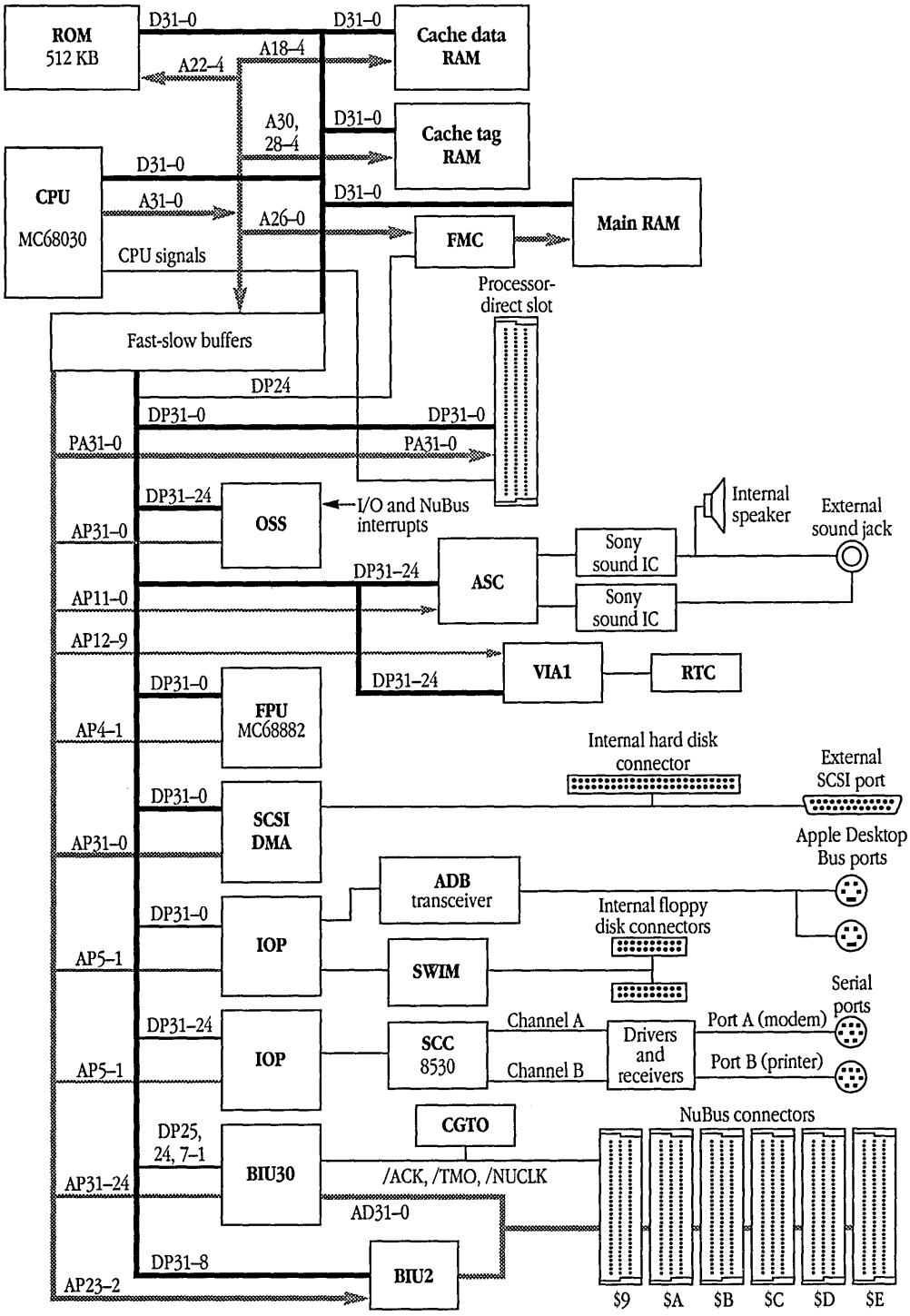
■ **Figure 1-2** Block diagram of the Macintosh IIx and Macintosh IIcx



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



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



RAM

RAM is the working memory of the system. Its base address is $00. The first 256 bytes of RAM (addresses $00 through $FF) 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.

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; this is different from the original Macintosh, Macintosh Plus, and Macintosh SE. Video RAM is located on the video card in a NuBus slot. (If on-board video is used in the Macintosh IIci, video RAM must be installed in RAM bank A on the main logic board.) The microprocessor has uninterrupted access to RAM, making the average RAM 32-bit access rate equal to 12.53 MB per second for the Macintosh II, and 15.6 MB per second for the Macintosh IIx and the Macintosh IIcx. In the Macintosh IIci, the use of burst mode and the RAM IC's fast-page mode give the processor a maximum RAM access rate of 36.36 MB per second. The average rate of the Macintosh IIci depends upon the amount of contention with the built-in video circuits. The average RAM access rate of the Macintosh IIfx is 64 MB per second based on successfully accessing the cache memory 90% of the time.

ROM

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

Device I/O

Computers in the Macintosh II 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 in 24-bit mode. New applications can take advantage of the full 32-bit mode for slot access as explained in Chapter 7, "NuBus Card Memory Access."

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, IIx, and IIfx, $9 through $B for the Macintosh IIcx, and $C through $E for the Macintosh IIci).

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

In general, ROM routines won't run in 32-bit mode under the current Macintosh Operating System. An application or driver written to run in 32-bit mode must switch to 24-bit mode before calling any ROM routine and return to 32-bit mode thereafter.

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. This call is described in the Operating System Utilities chapter of *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 the A/UX® operating system runs entirely in 32-bit mode. The MC68851 PMMU is also capable of ignoring the high eight bits of the address in order for the Macintosh Operating System to run in 24-bit mode.

The other computers in the Macintosh II family do not need an AMU or a PMMU because the MC68030 microprocessor used in these computers includes a built-in memory management unit that provides all necessary memory management functions, including 24-bit to 32-bit memory mapping translation and A/UX operating system support.

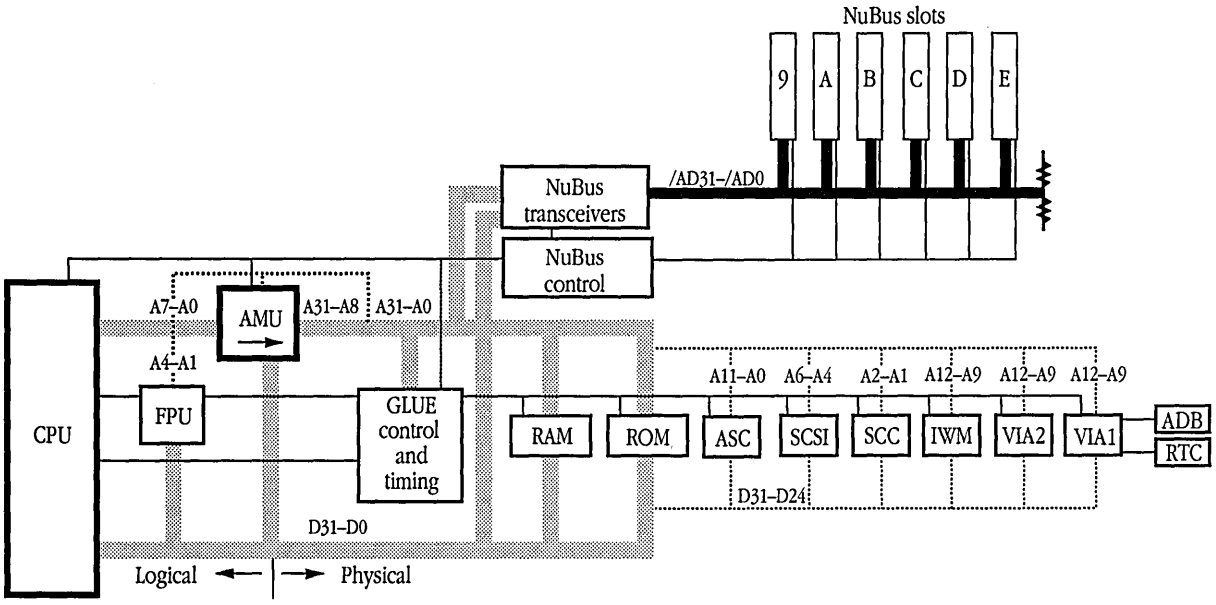
Chapter 7, "NuBus Card Memory Access," shows in detail how cards installed in NuBus slots address memory.

Address/data bus

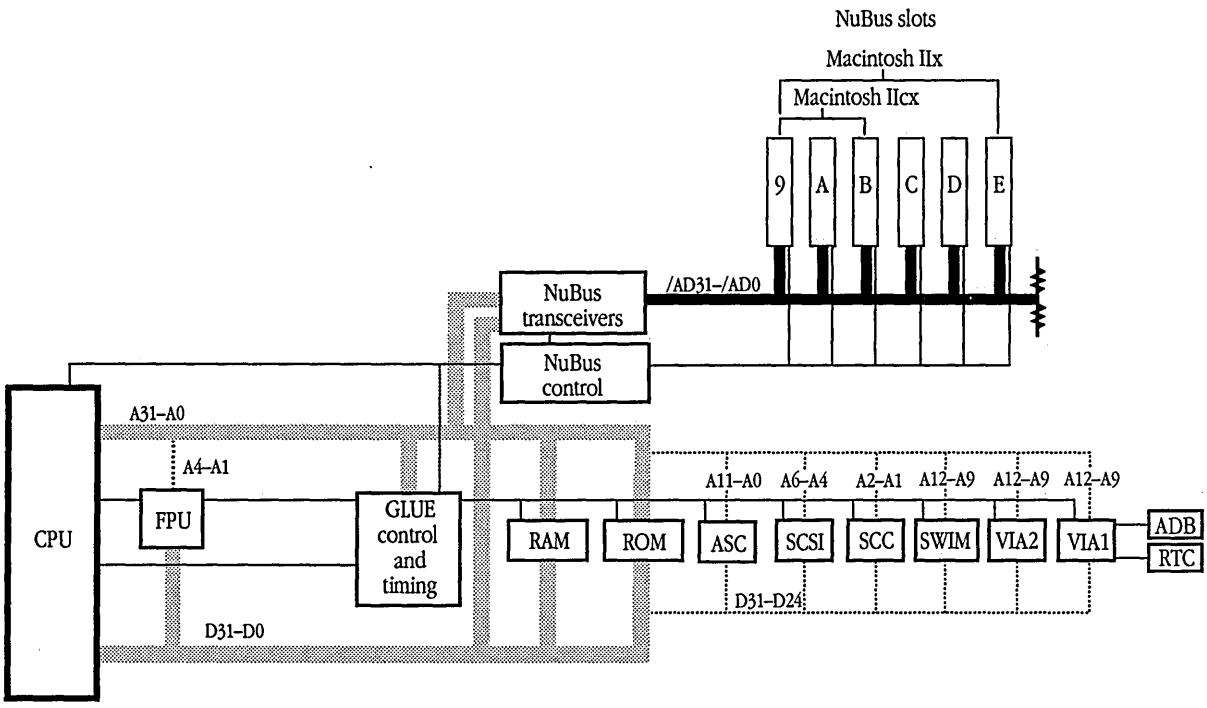
Figures 1-5 and 1-6 are examples showing the basic address/data bus architecture used in the Macintosh II, Macintosh IIx, and Macintosh IIcx computers. (Although examples of the Macintosh IIci and the Macintosh IIfx are not included, the function of the address/data bus interface in these machines is essentially the same.) Note that the address and data buses shown in Figures 1-5 and 1-6 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. Of primary interest in these figures are the address and data paths (shown grayed) to and from the NuBus transceivers. The utility, control, arbitration, and slot ID signals are listed in Table 2-2.

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.

■ **Figure 1-5** Macintosh II address/data bus architecture



■ **Figure 1-6** Macintosh IIx and Macintosh IIcx address/data bus architecture



NuBus interface architecture

All Macintosh II–family computers use a similar interface architecture to communicate with the NuBus. The most noticeable difference is that the newer computers, the Macintosh IIci and the Macintosh IIfx, have incorporated different NuBus control and data transceivers into their bus interface design. The Macintosh II, the Macintosh IIx, and the Macintosh IIcx all use the NuChip custom IC for their bus interface control function as shown Figures 1-1 and 1-2. In the Macintosh IIci (Figure 1-3), the bus interface control function is replaced by a NuChip 30 custom IC, and in the Macintosh IIfx (Figure 1-4), two custom ICs, the BIU30 and the BIU2, perform the control and transceiver functions. This section shows how the processor uses the bus interface control and transceiver logic to communicate with the NuBus.

The NuBus interface in Figures 1-1 through 1-4 shows bidirectional bus interface blocks between the microprocessor bus and the NuBus. Figure 1-7 shows a further breakdown of the functional elements comprising the bus interface circuits.

The bus interface control function is implemented as four state machines, three of which are shown in Figure 1-7. 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 microseconds). 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 1-7) 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 1-8). 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 timeout 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.

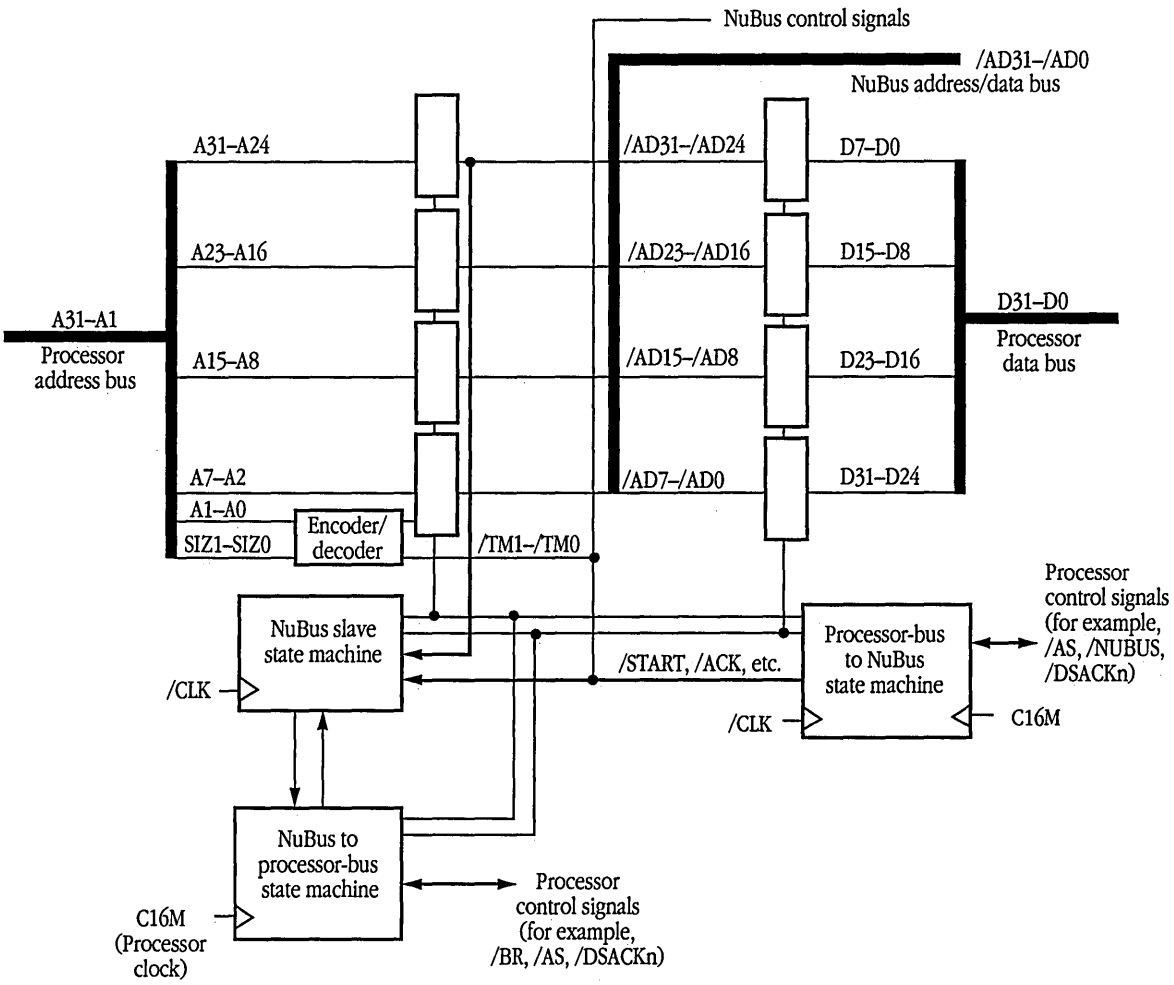
NuBus to processor-bus state machines

Two state machines, the NuBus slave and the NuBus to processor-bus, control a NuBus to processor-bus data transfer (see Figure 1-7).

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 $4x80 0000 through $4xFF FFFF on the processor bus is made to the ROM (see the map in Figure 1-8). 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. ▲

■ **Figure 1-7** Bus interface architecture

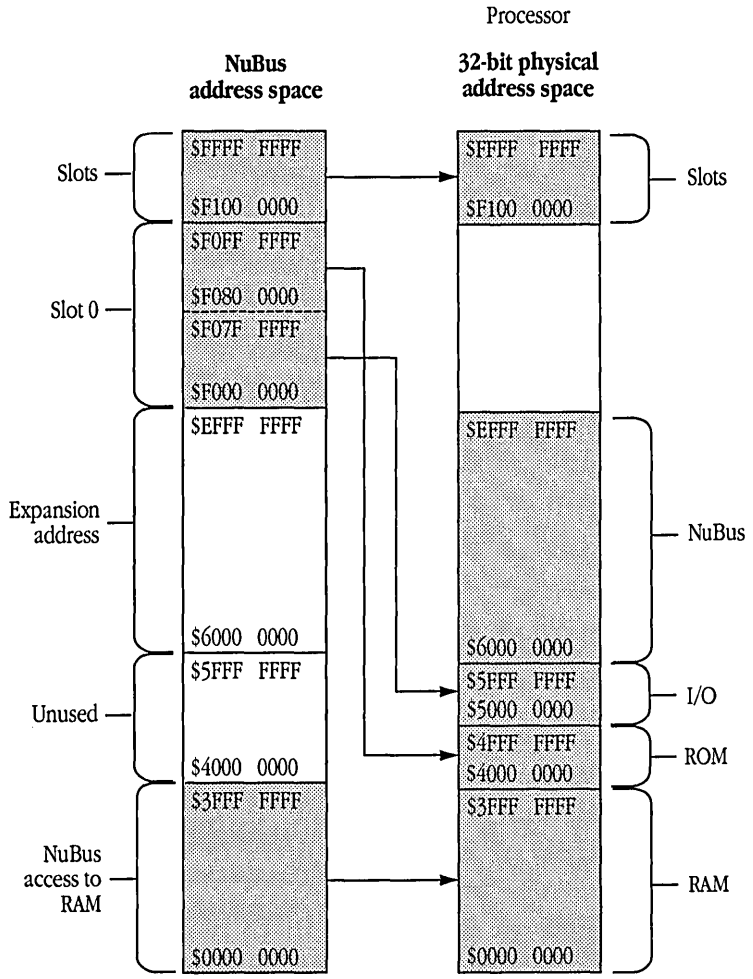


The Macintosh II, the Macintosh IIx, and the Macintosh IIcx have only 256 KB of ROM; this many locations require only 18 bits of addressing to specify a ROM location. The hardware decode logic interprets any physical address whose upper four address bits (A31–A28) are equal to $4 as a ROM access. So 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. 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 1-8** NuBus to processor-bus translation



Chapter 2 **NuBus Overview**

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.

NuBus features

The NuBus is used for expansion of a Macintosh II–family 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 computers in the Macintosh II family. 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.

■ **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 available but not implemented in the Macintosh II family).
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.

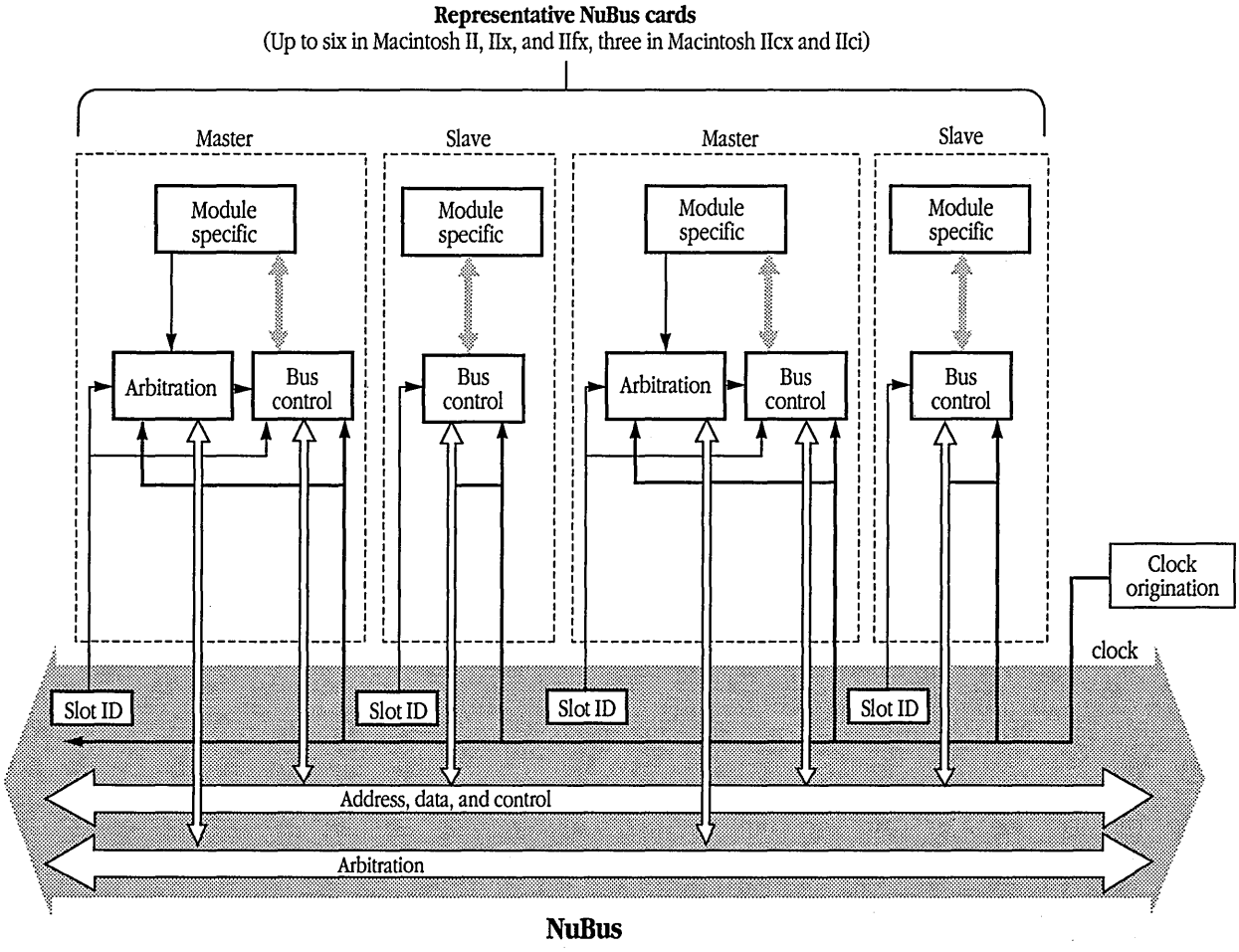
Figure 2-1 is a simplified representation of a typical NuBus system. In addition to the slot identification (ID), clock, address/data, and arbitration lines shown in the diagram, there are system reset, parity, power fail warning, non-master request, and data transfer control lines.

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 II family, however, interrupts are detected through the non-master 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 signal classifications

NuBus signals can be grouped into six classes based on the functions that they perform. There are also power and ground lines. Table 2-2 shows the NuBus signal classifications.

■ **Table 2-2** Classes of NuBus signals

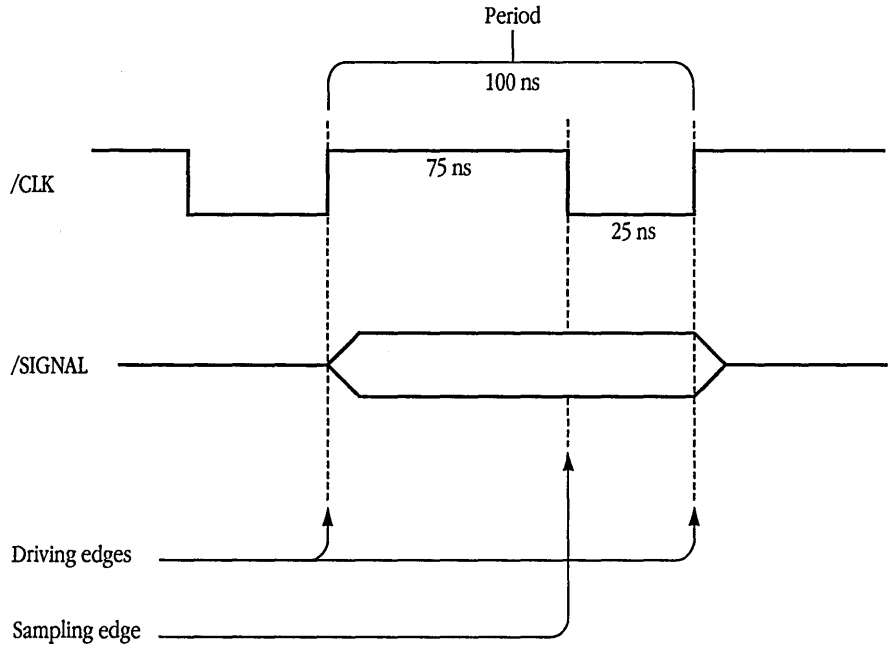
Classification	Signal	Signal description	Number of pins
Utility	/RESET	Reset	1
	/CLK	Clock	1
	/PFW	Power Fail Warning	1
	/NMRQ	Non-Master Request	1
Control	/START	Start	1
	/ACK	Acknowledge	1
	/TM0	Transfer Mode 0	1
	/TM1	Transfer Mode 1	1
Address/data	/AD31–/AD0	Address/Data	32
Arbitration	/ARB3–/ARB0	Arbitration	4
	/RQST	Request	1
Parity	/SP	System Parity	1
	/SPV	System Parity Valid	1
Slot ID	/ID3–/ID0	Slot Identification	4
		Total signals	**51**
Power/ground	+5V		11
	+12V		2
	–12V		2
	–5V (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.

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



Definitions

Table 2-3 defines terms used throughout Part I that are used to describe the NuBus expansion interface. The relationships among 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 2-3** Basic definitions

Term	Definition
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. The arbitration contest requires two bus periods (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 two types, attention-null and attention-resource-lock 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.
Card	A printed circuit board connected to the bus in parallel with other cards.

(Continued)

■ **Table 2-3** Basic definitions (Continued)

Term	Definition
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.
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. (Block data transfer is not implemented in the Macintosh II family.) See Figure 2-3.
Deasserted	For an active-low signal, synonymous with *high, inactive, unasserted, false,* and *released.*
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.*
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.
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 2-3** Basic definitions (Continued)

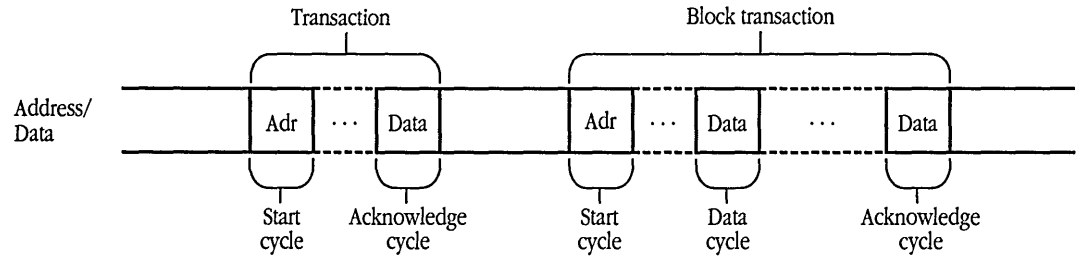
Term	Definition
Released	For an active-low signal, synonymous with *high, inactive, deasserted, unasserted,* and *false.*
Sampling edge	The falling edge (high to low) of the central system clock (/CLK). See Figure 2-2.
Slave	A card that responds to being addressed by another card acting as a master. The main logic board in the Macintosh II family 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.
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 IIcx and Macintosh IIci have three slots).
Slot ID	The hex number ($9 through $E in the Macintosh II, Macintosh IIx, and Macintosh IIfx; $9 through $B in the Macintosh IIcx; and $C through $E in the Macintosh IIci) 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.
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 numbers.
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	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 is assigned to the card in slot *s,* where *s* is an ID digit in the range $9 through $E.

(Continued)

■ **Table 2-3** Basic definitions (Continued)

Term	Definition
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."
Three-state	A bus driver that drives a line low or high or doesn't drive it at all. Also commonly called *tri-state.*
Transaction	A complete NuBus operation such as read or write. In the Macintosh II family, 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 acknowledgement information. See Figure 2-3.
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 and is 32 bits long; a halfword is 16 bits long (usage is consistent with the Texas Instruments NuBus specification). The data type word, however, is 16 bits long (see Table 8-1); 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.

■ **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.

Utility signals

This section identifies the signal lines that serve utility functions for the NuBus interface. The main logic board of a Macintosh II–family computer provides the structure and the slot connectors; it also provides the Clock and Reset signal sources and bus timeout circuitry.

Clock signal

Clock (/CLK), driven from a single source, synchronizes bus arbitration and data transfers between system cards. /CLK 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.

Reset signal

Reset (/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:

1. **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.
2. **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.
3. **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

Power Fail Warning (/PFW) may be asserted asynchronous with respect to the driving edge of /CLK and indicates that the power is about to fail. In the Macintosh II family, 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.

Non-Master Request signal

Non-Master Request (/NMRQ) is a signal asynchronous to /CLK that provides an interrupt mechanism for cards that are intended to be slave-only. Such cards avoid the cost of implementing arbitration logic.

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 IIcx and Macintosh IIci have only three slots numbered $9 through $B and $C through $E, respectively. 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 kilohm pull-up resistors; you should, however, be able to use resistors as high as 10 kilohms 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 1/16th (256 MB) of the entire 4 gigabyte NuBus address space is called the **standard slot space.** If /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 (1/16th of 256 MB) per NuBus card slot, an address region called a **slot.**

However, by using the /ID3–/ID0 bits in a different way, a second natural address decode of what is called **super slot space** can be easily performed. If /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 (1/16th 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 clock cycle n, then it is determinate during 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 three-state signal is asserted during cycle n, and is not driven during cycles n + 1 and n + 2, then it is *not* guaranteed determinate during cycle n + 1, but the line 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 three-state.

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. /START 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.

Transfer Mode 0 and 1 (/TM0, /TM1) 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 acknowledgement. /TM0 and /TM1 encoding for start cycles is given in Table 3-1.

Address/Data signals

Address/Data 0 through 31 (/AD31–/AD0) signals 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 /TM0 and /TM1, carry transfer mode information during the start cycle. This transfer mode encoding is shown in Table 3-1.

Bus parity signals

System Parity (/SP) transmits parity information between NuBus cards that implement NuBus parity checking. Future Apple products may employ this feature, but the Macintosh II family does not provide bus parity checking, so this line is pulled high.

System Parity Valid (/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 II family, 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.

Data transfer specifications

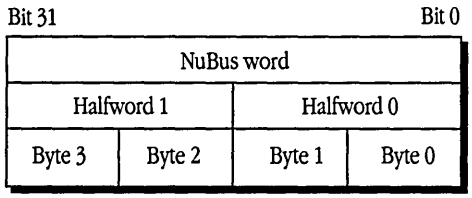
The NuBus supports reads and writes of several different data sizes. Although optimized for transactions of words and blocks of words, the NuBus also supports byte and NuBus halfword transactions as shown in Figure 3-1. The base unit of addressability is a NuBus word; /AD31–/AD2 specify the appropriate word. The two least significant address bits (/AD1–/AD0), along with /TM1–/TM0, specify the transfer mode; that mode determines which part of the addressed word is to be transferred, as shown in Table 3-1.

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.

■ **Table 3-1** Transfer mode coding

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

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



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 not supported in the current members of the Macintosh II family, but they may be used in future products. 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 slaves sample the /ADx and /TMx lines.

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

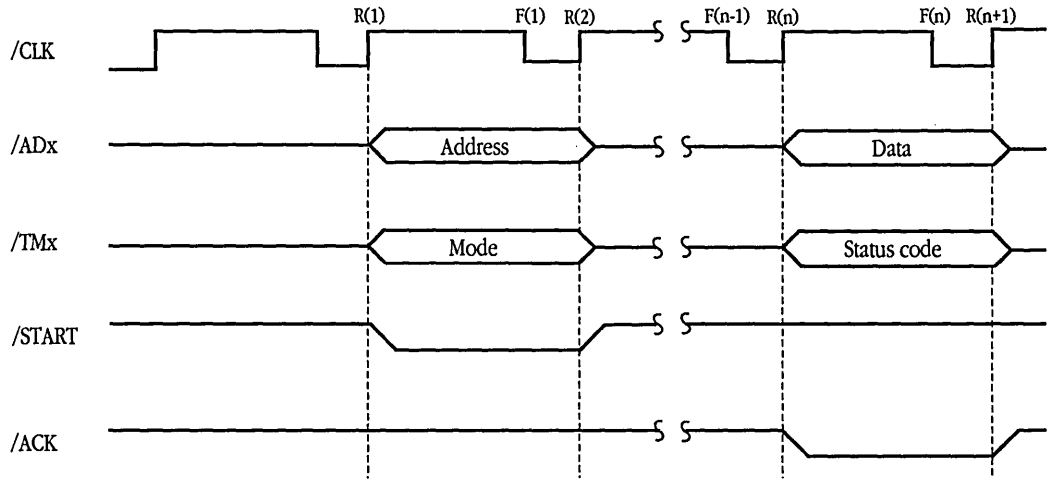
R(n)§ The bus slave drives the requested data onto the /ADx lines, drives the appropriate status code on /TM0 and /TM1, and drives /ACK low.

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.
§ $2 \leq n < 256$, the system defined timeout period.

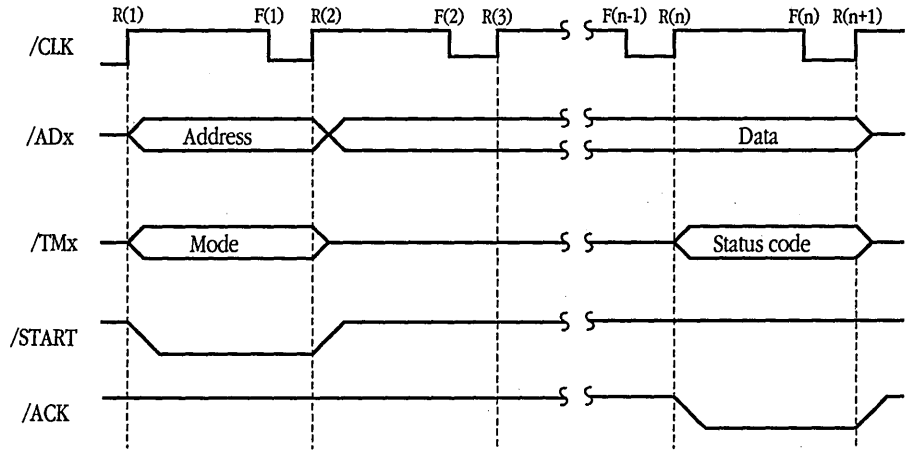
■ **Figure 3-2** Timing of NuBus read transaction



Write transactions

Figure 3-3 shows the timing for write operations other than block transfers. Block transfers are not supported in current Macintosh II–family computers, but they may be used in future products. 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. (See Figure 7-2.)

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 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.
§ $2 \leq n < 256$, the system defined timeout 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 timeout 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 timeout: 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 timeout error response. This timeout response indicates that the system defined timeout 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 timeout support logic on the Macintosh main logic board enforces a period of 256 clock periods, or 25.6 microseconds, and assumes the role of the nonresponding slave; it generates an acknowledge cycle with a bus timeout 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 computer in the Macintosh II family generates a processor-bus error exception (/BERR signal) if its microprocessor attempts a NuBus access that is terminated with an error, bus timeout, or 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. Two of the four available codings are used at present, as 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	Reserved

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

Attention cycles are used to reinitiate bus arbitration (attention-null) or 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

Attention-resource-lock: /TM1 high (H) and /TM0 low (L) signal an attention-resource-lock cycle 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.

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 II family of computers.

By write transaction

Interrupts on the NuBus can be implemented as write transactions. *Interrupts are not done this way on Macintosh II-family 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 (Non-Master 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 II- family computers.*

By a dedicated /NMRQ line from each slot (Macintosh II-family computers)

Computers in the Macintosh II family use a separate (non-NuBus) /NMRQ line from each slot to support interrupts (see Figures 1-1 through 1-4). 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.

Block data transfers

Block transfers are not implemented in Macintosh II-family computers, but they may be implemented in future Apple products. The following discussion, although not pertinent to the Macintosh, is provided for completeness in describing the NuBus.

Block transfers are transactions that consist 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 master and communicated during the start cycle. Allowed lengths of block transfers are 2, 4, 8, and 16 words. (Only 32-bit NuBus word transfers are supported in block transfer mode.)

The /TM1, /TM0, /AD1, /AD0 encoding for 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 block transfers, 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 acknowledgements, /TM0 has the same significance and timing as the /ACK signal for nonblock transfers. The acknowledgement of the final word transfer is a standard acknowledge cycle. Status codes are shown in Table 3-2.

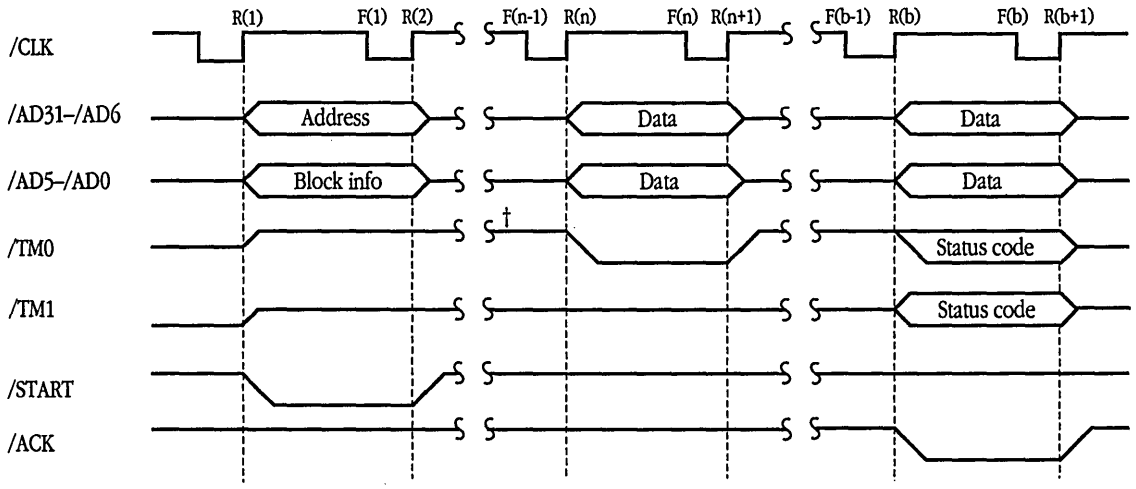
■ **Table 3-4** Block size and starting address coding

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

Block read

Figure 3-4 shows the timing for a NuBus block read transaction. See Table 3-1 for the /TM1, /TM0, /AD1, /AD0 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 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 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 slaves sample 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. /TM0 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.

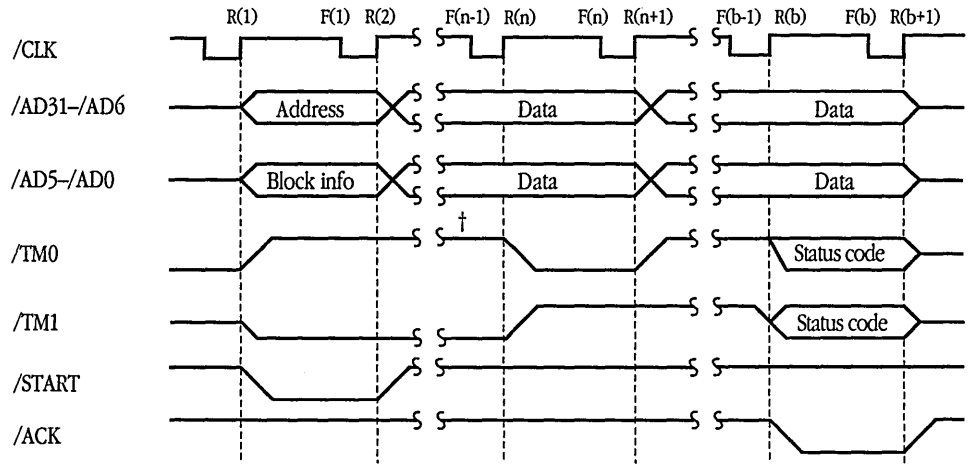
R(b + 1) The bus slave releases the /ADx, /ACK, and /TMx lines.

† R is the rising edge of /CLK.
‡ F is the falling edge of /CLK.
§ 2 ≤ n ≤ 256, the system defined timeout period.
¶ 2 ≤ b ≤ 256B.

Block write

Figure 3-5 is a timing diagram for a NuBus 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 for NuBus 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 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 slaves sample 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 an intermediate acknowledge (/TM0 asserted).

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

F(n–) The bus slave samples the /ADx lines to capture the data being written. The n– notation implies 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.
§ $2 \leq n \leq 256$, the system defined timeout period.
¶ $2 \leq b \leq 256B$.

Block transfer errors

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

The speed of a block transfer is controlled by the slave; therefore, a master requesting a 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 block transfer, it must support all types of data transfer (byte, halfword, and word). In the case of a 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.

Nonaligned microprocessor accesses

The MC68020 and MC68030 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 II–family computer and NuBus always translates an aligned request into its counterpart on NuBus. However, that interface also provides support for all nonaligned microprocessor accesses (by using the dynamic bus sizing facilities of the microprocessor). 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 provides the required data in the fewest NuBus transactions. Notice that 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 one byte to $Fs00 0004, which is a natural NuBus transaction.

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 the newest Macintosh II–family 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 four 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. 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 or MC68030 microprocessors on the main logic board of a Macintosh II–family 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 in width, 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 computer in the Macintosh II family properly responds to these two requests. The same instruction when executed by the MC68020 or MC68030 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.

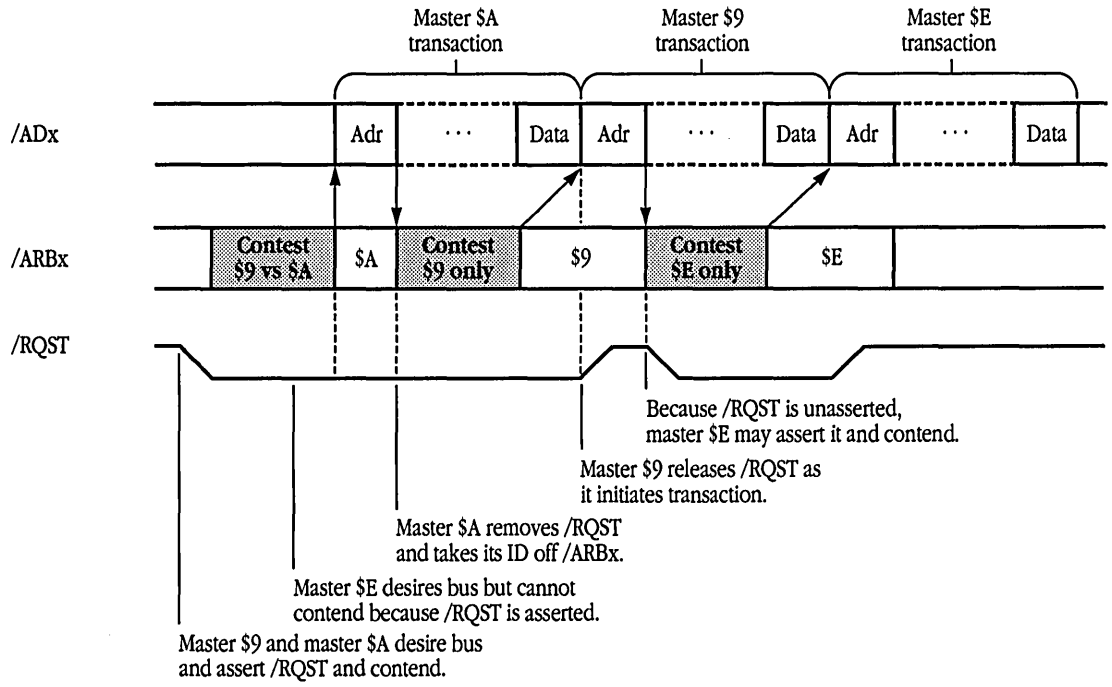
During arbitration, one or more cards contend for control of the NuBus. Cards that desire ownership of the NuBus must first assert the /RQST line. /RQST may be asserted only while it is in an unasserted state. All cards that assert /RQST place their ID codes on the /ARBx lines and contend for the bus. 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 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 both wins a contest and the transaction in progress ends.

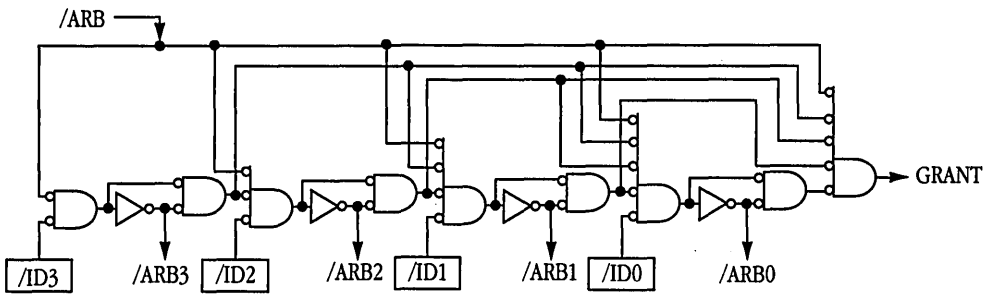
■ **Figure 4-1** Sample arbitration contest



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 B 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.

Locking

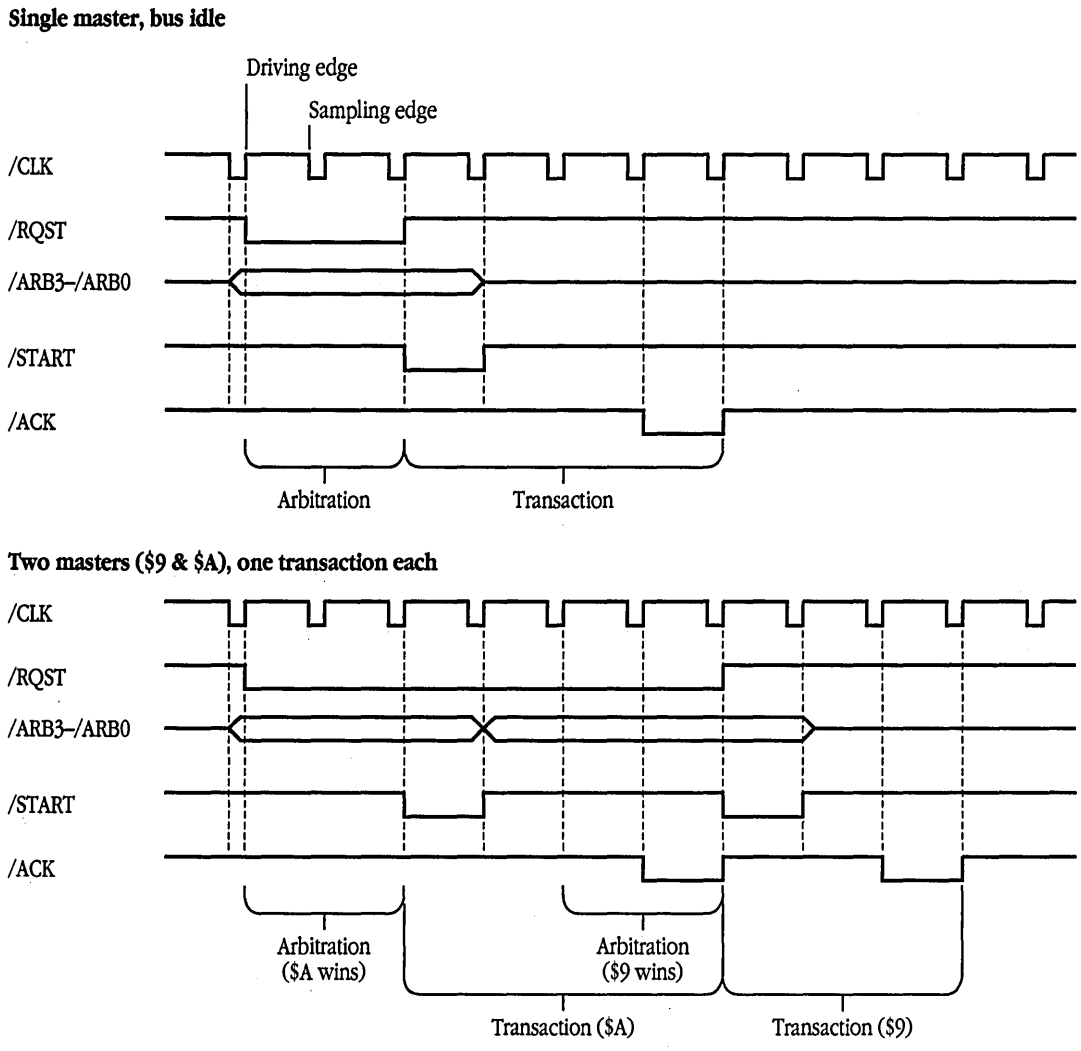
Although cards generally use the bus for a single transaction before allowing another requesting card to become bus master, sometimes the bus must be held locked in an extended tenure. For some local processor operations, it may be necessary to prevent any NuBus requests from interfering with the access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

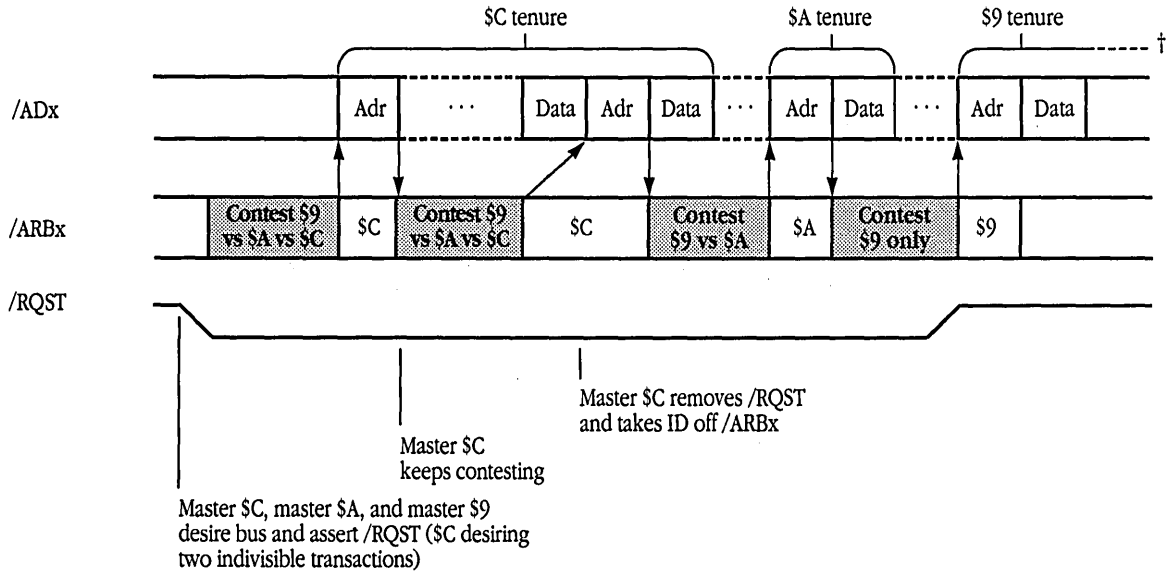


Bus locking

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.

■ **Figure 4-4** Sample bus lock



† 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.

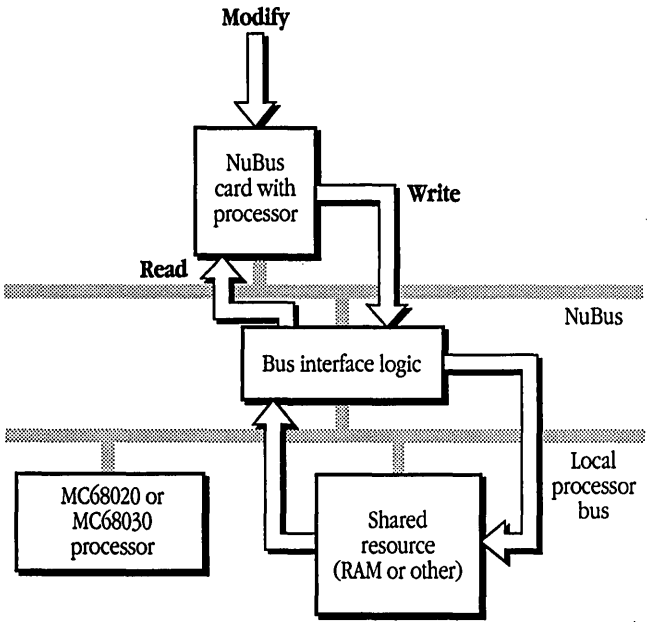
Access to a resource must be controlled when that resource is accessible by both a local processor and the NuBus. One example of such a shared resource is a dual-ported RAM. Another, more specific, example is found in Macintosh II–family computers, where the NuBus interface circuitry uses the local processor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

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



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 requestor, 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)	> 2.0V at the receiver
L (asserted)	< 0.8V 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)
- power control (/PFW)

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

† Supplied by the Macintosh II-family 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 tri-stated 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 volts for at least 1.5 seconds, the power supply turns on and the computer begins operating. Once the power supply turns on, its own +5 volt output holds /PFW high so it can continue operating. If /PFW is pulled below .6 volts, the power supply will turn off; /PFW should be *held* below .6 volts 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 +5-volt 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 +5-volt (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 pullup may be added to the /PFW line or else Q4 on the main logic board may not be able to turn 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. The order of the rows is given as viewed from the front edge of the card.

■ **Table 5-3** 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	‡	GND	‡	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	†	/AD4	24	/ARB1	†	/ARB0
9	/AD7	†	/AD6	25	/ARB3	†	/ARB2
10	/AD9	†	/AD8	26	/ID1	†	/ID0
11	/AD11	†	/AD10	27	/ID3	†	/ID2
12	/AD13	GND	/AD12	28	/ACK	+5V	/START
13	/AD15	GND	/AD14	29	+5	+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

† 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 standard IEEE 1196; in the Macintosh II family, they are grounded.

Power supply specifications

Three voltages are specified on the NuBus: +5 V, +12 V, and –12 V. These voltages are listed in Table 5-4 with their specifications.

■ **Table 5-4** Power supply specifications

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

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 II–family computer, with equal current allocation to each of the slots, yields the recommendations in Table 5-5. 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 5-5.

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

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

◆ *Note:* The current analysis assumed 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.

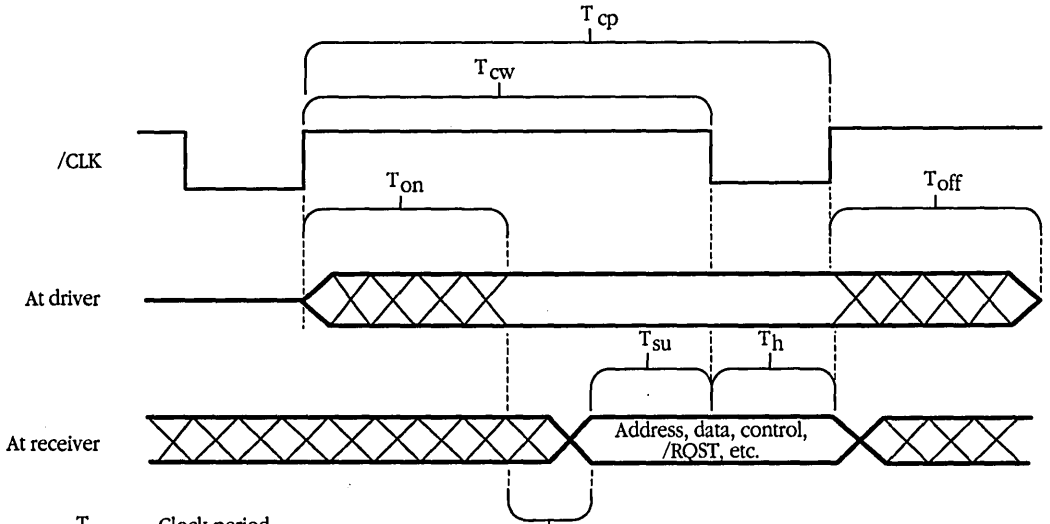
Timing requirements

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 II–family computer 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 5-6 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 propagation

■ **Table 5-6** Data transfer timing parameters

Parameter	Description	Minimum	Maximum	Units
T_{cp}	Clock period	99.99	100.01	ns
T_{cw}	Clock width	73	77	ns
T_{on}	Turn-on time	0	35	ns
T_{off}	Turn-off time	0	35	ns
$2T_{pd}$	NuBus delay	—	17	ns
T_{su}	Setup time	21	—	ns
T_h	Hold time	T_{cp}–T_{cw}	—	ns

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

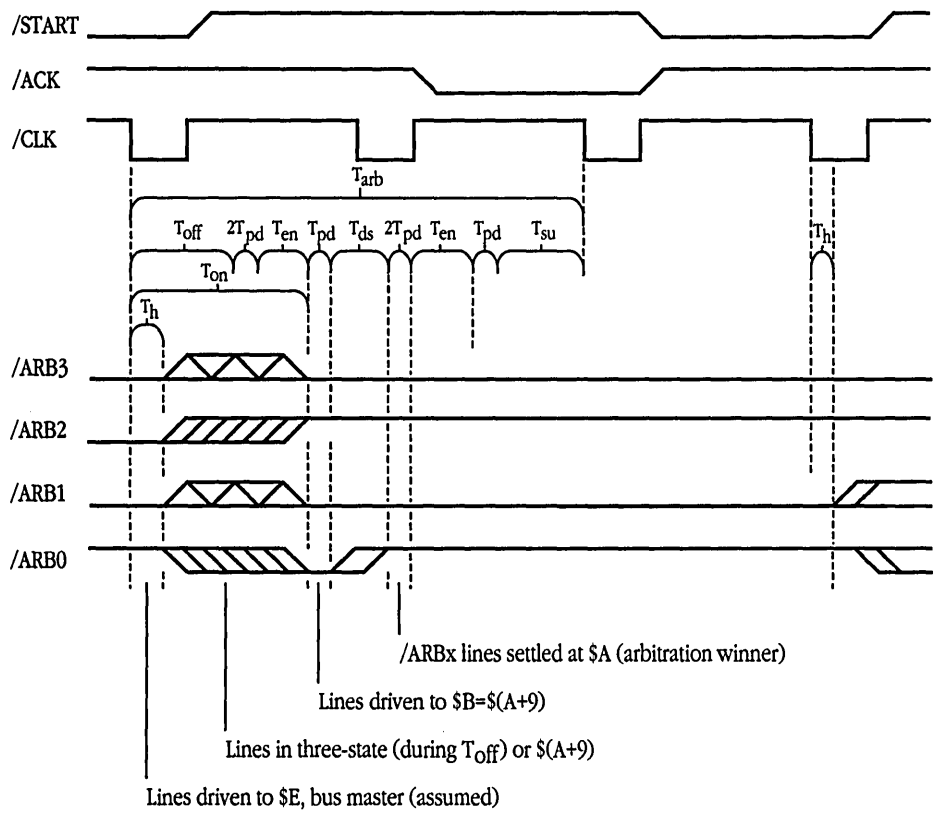
Refer to Chapter 4, "NuBus Arbitration," for a description of the arbitration process. The timing for the /ARBx signals is not the same as the timing of the data transfer signals. Arbitration begins on the falling (sampling) edge of /CLK before the assertion of /RQST or, if /RQST is already active on 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 5-7 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 5-7 lists the timing specifications for the /ARBx lines.

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



■ **Table 5-7** Bus arbitration timing parameters

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

Chapter 6 **NuBus Card Physical Design Guide**

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

- *Note:* The Texas Instruments NuBus documentation also specifies a much larger, triple-height card, but that card cannot be used in a Macintosh II–family computer.

Card description

Three foldout drawings in the back of the book show the pertinent design details and installation requirements of a NuBus expansion card. Foldout 1 shows the overall dimensions and the placement of connectors on a typical NuBus card. Foldout 2 gives the clearance dimensions for installing a NuBus card in a Macintosh II, Macintosh IIx, or Macintosh IIfx. Foldout 3 gives the clearance dimensions for installing a NuBus card in a Macintosh IIcx or Macintosh IIci.

▲ **Warning** Foldouts 1 through 3 are from design guides used within Apple Computer. These drawings were correct at the time of publication but are subject to future change. ▲

△ **Important** To make sure that your NuBus card fits and functions in all Macintosh II–family computers, your physical design should adhere to the specifications in the Institute of Electrical and Electronic Engineers publication *IEEE Standard for a Simple 32-Bit Backplane Bus: NuBus, ANSI/IEEE Std 1196-1987.* △

A NuBus expansion card must be 4.0 inches in height and between 12.875 and 7.0 inches in length. Foldout 1 shows a card viewed from the component side. The NuBus connector is on the bottom edge of the card in the drawing. The I/O connector is on the right side.

Card thickness must be 0.062 ± 0.0075 inches. Warpage must be controlled to within 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 0.133 inches (3.38 mm) 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 0.60 inch, measured from the card surface. No component or wire lead is allowed to extend more than 0.10 inch beyond the noncomponent side of the card.

The nominal spacing between centerlines of adjacent NuBus connectors is 0.900 inches in the Macintosh II, Macintosh IIx, and Macintosh IIfx computers and 0.950 inches in the Macintosh IIcx and Macintosh IIci computers.

NuBus connector description

The NuBus connector on the card must be a 603-2-IEC-C096-M connector. Pin assignments are as shown earlier in Table 5-3. 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 the main logic board of a Macintosh II–family computer. Note that this is the same as the connector used in the Macintosh SE and shown in Figure 15-6.

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.

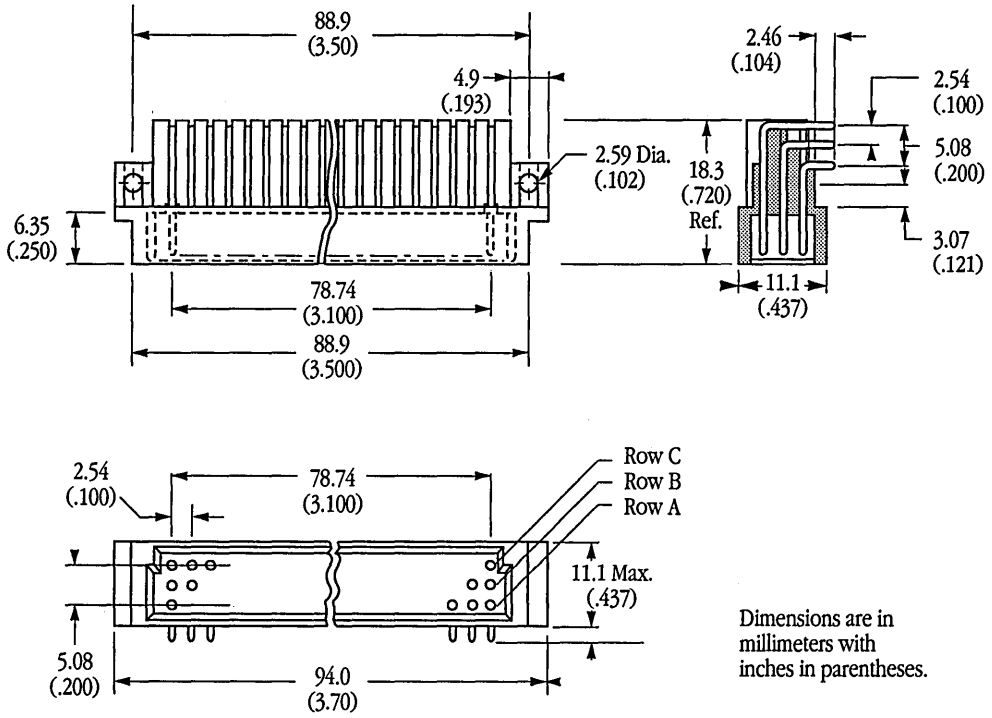
For EMI protection, a metal shield surrounds the I/O connector on the rear of the card. Appendix A, "EMI, Heat Dissipation, and Product Safety Guidelines," provides information on EMI reduction when a Macintosh II–family computer is expanded. See Foldout 4 at the back of the book for a drawing of the I/O connector shield.

▲ **Warning** Foldout 4 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to future change. ▲

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

An auxiliary connector is allowable, but discouraged. It must be no longer than 3.0 inches and be located as shown in *IEEE Standard for a Simple 32-Bit Backplane Bus: NuBus, ANSI/IEEE Std 1196-198.*

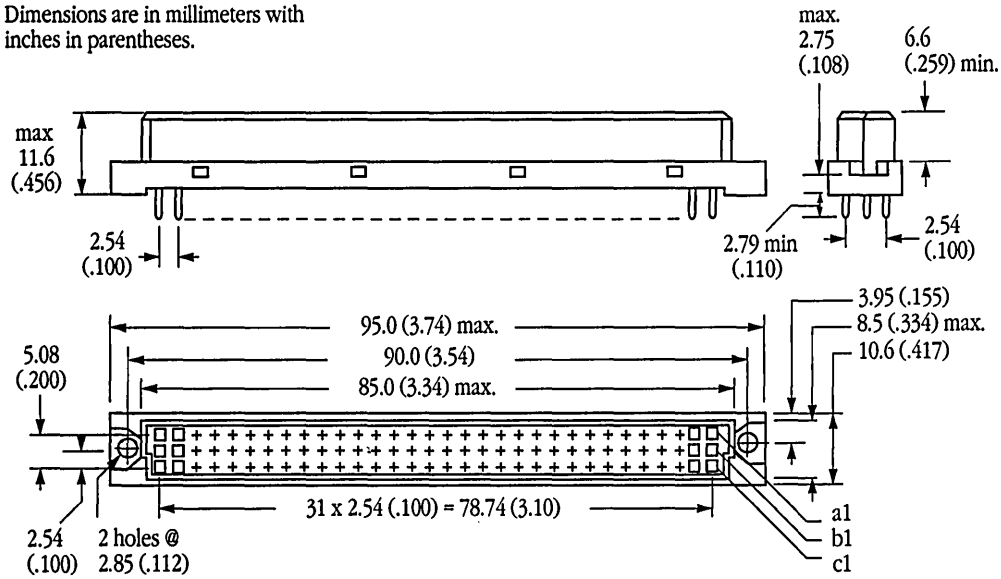
■ **Figure 6-1** 96-pin plug connector for a Macintosh II–family NuBus expansion card



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

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

Dimensions are in millimeters with inches in parentheses.



Three-row socket connector

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

Recommended heat dissipation guidelines

Apple recommends that Macintosh II–family expansion cards should dissipate a maximum of 13.3 watts 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 watts 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 watts 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.

Third-party design aids

There are a number of products available that will help you design your expansion cards much faster and more easily. For example, you can purchase blank NuBus expansion cards for prototyping from

Diversified I/O, Inc.
1008 Stewart Drive
Sunnyvale, CA 94086
(408) 730-2171
AppleLink: D0242

Also, you can quickly implement the NuBus interface circuitry by using a VLSI (very large scale integration) chip set designed specifically for this purpose. Currently there are two VLSI chip sets available. These are marketed by Texas Instruments, Inc. and Pinnacle Micro, Inc.

For information on the Texas Instruments NuBus chip set, contact

Texas Instruments NuBus Marketing
Texas Instruments, Inc.
P.O. Box 655012
Dallas, TX 75265
(214) 997-5499

For information on the Pinnacle Micro NuChips, contact

Pinnacle Micro, Inc.
15265 Alton Parkway
Irvine, CA 92718
(800) 553-7070
FAX: (714) 727-1913
AppleLink®: D2064

Chapter 7 **NuBus Card Memory Access**

This chapter describes how cards connected to a Macintosh II–family 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 II–family 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."

Address space

The NuBus architecture allows full 32-bit addresses, providing four gigabytes of address space. The upper one-sixteenth (256 megabytes) 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 megabytes 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 the Macintosh II family uses only slot IDs $9 through $E, only the six standard slot spaces $F9xx xxxx through $FExx xxxx are actually used.

◆ *Note:* For convenience, this section refers only to a NuBus configuration of six slots represented by slot IDs $9 through $E. Keep in mind that the Macintosh IIcx, with its three NuBus slots, uses slot IDs $9 through $B and has only three slot spaces, $F9xx xxxx through $FBxx xxxx. The Macintosh IIci also has only three NuBus slots, uses slot IDs $C through $E, and has only three slot spaces, $FCxx xxxx through $FExx xxxx.

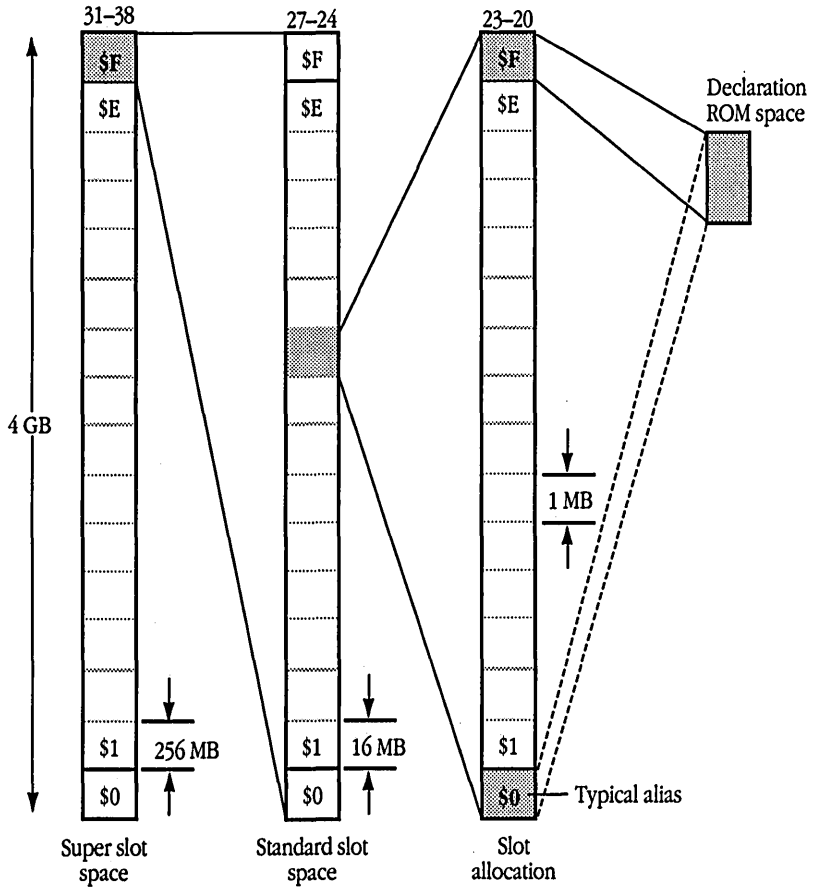
This system of fixed address allocations, based solely on a card's slot location, makes it possible for you to design cards that are free of jumpers and configuration switches.

▲ **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 megabytes, 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 megabytes 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.

■ **Figure 7-1** NuBus address space



As explained in *Inside Macintosh,* a Macintosh II–family 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 one megabyte of each card's standard 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 one megabyte 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 7-1.

■ **Table 7-1** 24-bit 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	

Macintosh II–family address allocations

All of the existing Macintosh II–family address space is accessible from NuBus. It is mapped onto the NuBus address space as shown in Table 7-2.

When the microprocessor accesses 32-bit addresses in the range $6000 0000 through $FFFF FFFF (except for $F0xx xxxx), it initiates a NuBus transaction. The mapping shown in Table 7-2 is correct for current members of the Macintosh II family. Future Macintosh products may have different mappings.

■ **Table 7-2** NuBus to Macintosh II–family processor address mapping

24-bit addresses from processor	32-bit addresses from processor	NuBus addresses	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	Future RAM
$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 0000 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 on 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
$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	Presently unused
$xxs0 0000 to $xxsF FFFF	$Fs00 0000 to $FsFF FFFF or $Fs10 0000 to $FsFF FFFF	$Fs00 0000 to $FsFF FFFF or $Fs10 0000 to $FsFF FFFF	Standard slot space, slot s (*s* in the range $9–$E)
	$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 one megabyte of each card's standard slot space is mapped onto a part of the 24-bit Macintosh II 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 the Macintosh II–family computers in 32-bit mode and by cards in other slots. These allocations are listed in Table 7-3.

■ **Table 7-3** 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 megabytes of NuBus slot space from $F000 0000 to $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 or the MC68030 bus. To achieve byte addressing consistency, the Macintosh II–family computers perform byte swapping of data between the microprocessor and the NuBus. This section explains the rationale and details of this implementation.

The routes by which bytes are transferred between the NuBus and the Macintosh II–family microprocessor are called **byte lanes.** Each NuBus addressable byte has a particular byte lane in which it is transferred; all bytes with addresses of the form (xxx modulo 4) = N are transferred in byte lane N. 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 (msb), while in the 68020 and 68030 microprocessors, byte 3 is the least significant byte (lsb) of a longword (32-bit) value.

In designing the Macintosh II 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 four bytes within a NuBus word and a processor longword are connected to each other.

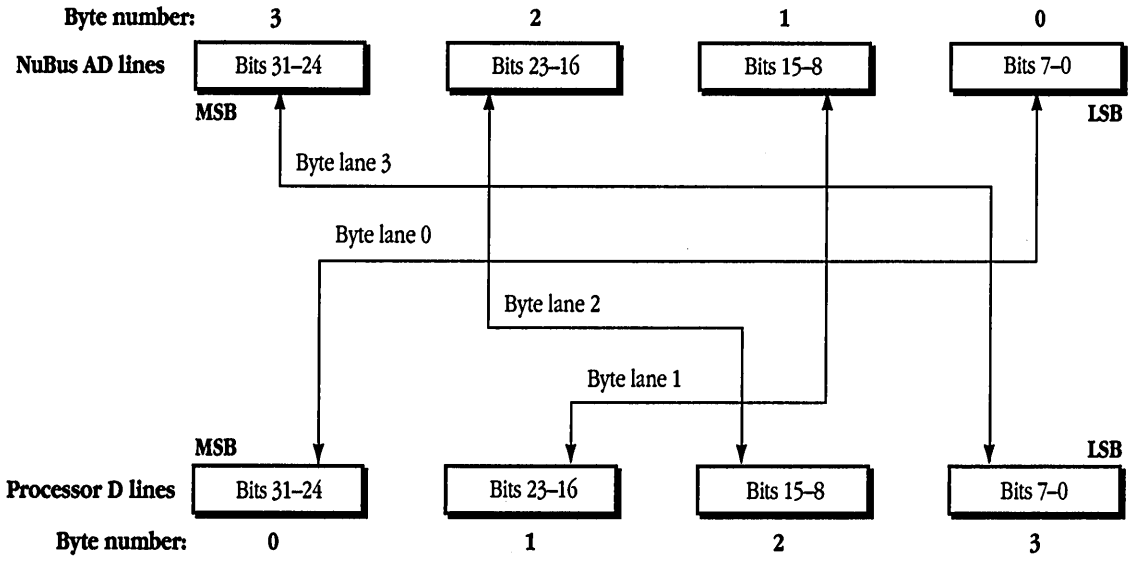
Apple chose to preserve byte address consistency in the Macintosh II family; each of the four bytes of the processor is connected to its corresponding NuBus byte lane. That is, byte *n* of the processor is connected to NuBus byte lane *n,* as shown in Figure 7-2. Byte lanes are numbered to reflect the 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 the Macintosh II–family 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 msb of a NuBus word is in byte lane 3, while the msb of a 68020 or 68030 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, for example, has byte ordering identical to NuBus; that is, the lsb of an 80386 word is byte 0, the msb is byte 3.

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

■ **Figure 7-2** Byte-lane mapping



Communication between a Macintosh II–family computer and a NuBus card may use any combination of one or more byte lanes. This subject is discussed in more detail under "The Format Block" in Chapter 8; possible byte-lane combinations are shown there in Table 8-2.

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.

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 II–family 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
- sample code for a typical NuBus card

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.

About the Slot Manager and the declaration ROM

The Slot Manager and declaration ROM have three main goals:

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

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

The **Slot Manager** is a group of routines in the ROM of a Macintosh computer that communicate with the declaration ROM firmware on a NuBus expansion card. 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 II–family 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 in the system heap that your application can access by using the Slot Manager routines. The Slot Manager chapter of *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 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.

About 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.** 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 usually provides information about that particular function to high-level applications that are interested in getting access to the function. 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 multi-function card that contains a parallel port, a serial port, and a modem. In this case, the card's declaration ROM would 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 and it 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 empty 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

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, you use the Slot Manager to find the beginning of the sResource, searching either by 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 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	Under a given category, you use the cType field to identify a sub-type of that category. For example, under Category Display, there might be cType entries such as Video and LCD. (For commonly predefined categories, 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. 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 (called *equates*) 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 equate 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.

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.

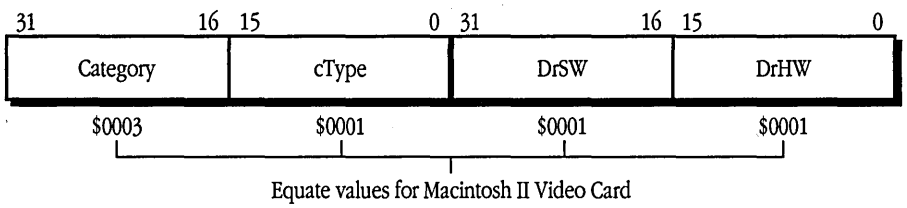
sRsrcType fields for a video card functional sResource

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 $0003000100010001. 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 only needs to assign it 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** Format and hierarchical structure of the sRsrcType fields for a video card functional sResource

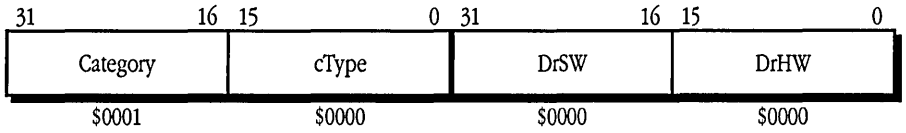


CatDisplay	Equate value of $0003
TypVideo	Equate value of $0001
DrSWApple	Equate value of $0001
DrHWTFB	Equate value of $0001
DrHWproductA	Equate value of $0002
DrHWproductB	Equate value of $0003
DrHWproductC	Equate value of $0004
DrHWproductD	Equate value of $0005
DrSWcompanyA	Equate value of $0002
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 equate 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** Format and hierarchical structure of the sRsrcType fields for a video card 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 manufacturer's 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 'INIT' file.

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

To obtain card identification and functional sRsrcType values, contact Apple Macintosh Developer Technical Support (MacDTS) with the following information:

- the functions the card performs
- the official product name (or code name) for the card
- the driver status (will the card have a software driver other than one that has been predefined, such as Apple's video driver?)
- the driver location (will the driver be on board in ROM or does it get installed at initialization time? is it 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. A HyperCard stack that facilitates entering and sending this information to MacDTS is available on the *Developer Helper* CD-ROM as well as on AppleLink.

Data types

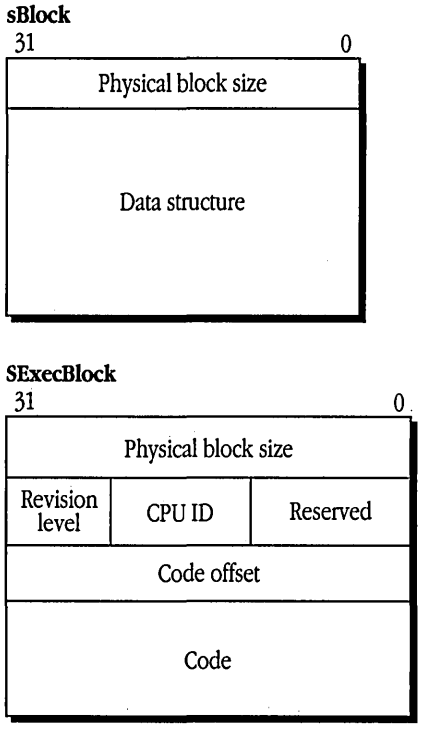
Table 8-1 shows the data types used for communication between the Slot Manager and the card's declaration ROM firmware. Two of the data types are illustrated in Figure 8-3.

■ **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 revision level field is always 02, the reserved field is always 00, and the CPU ID field identifies the processor—01 for the 68000, 02 for the 68020, and 03 for the 68030.

■ **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.

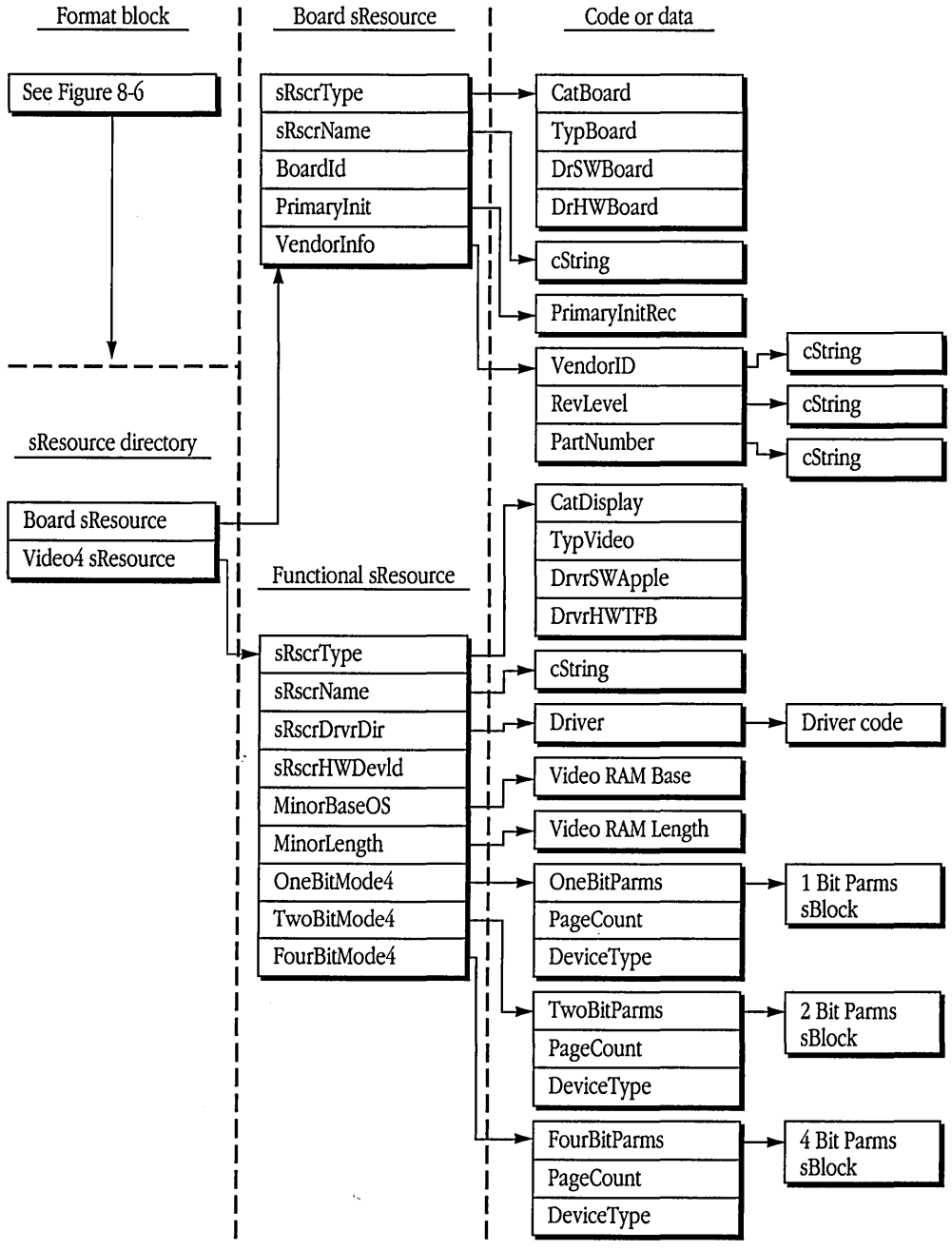
Video cards are more complex than other NuBus cards. They require additional elements in their firmware structure, which are described later in this chapter in the section "Additional Firmware Requirements of Video Cards."

Every NuBus card's declaration ROM firmware must include a format block, an sResource directory, and a Board sResource, and 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 at the end of this chapter.

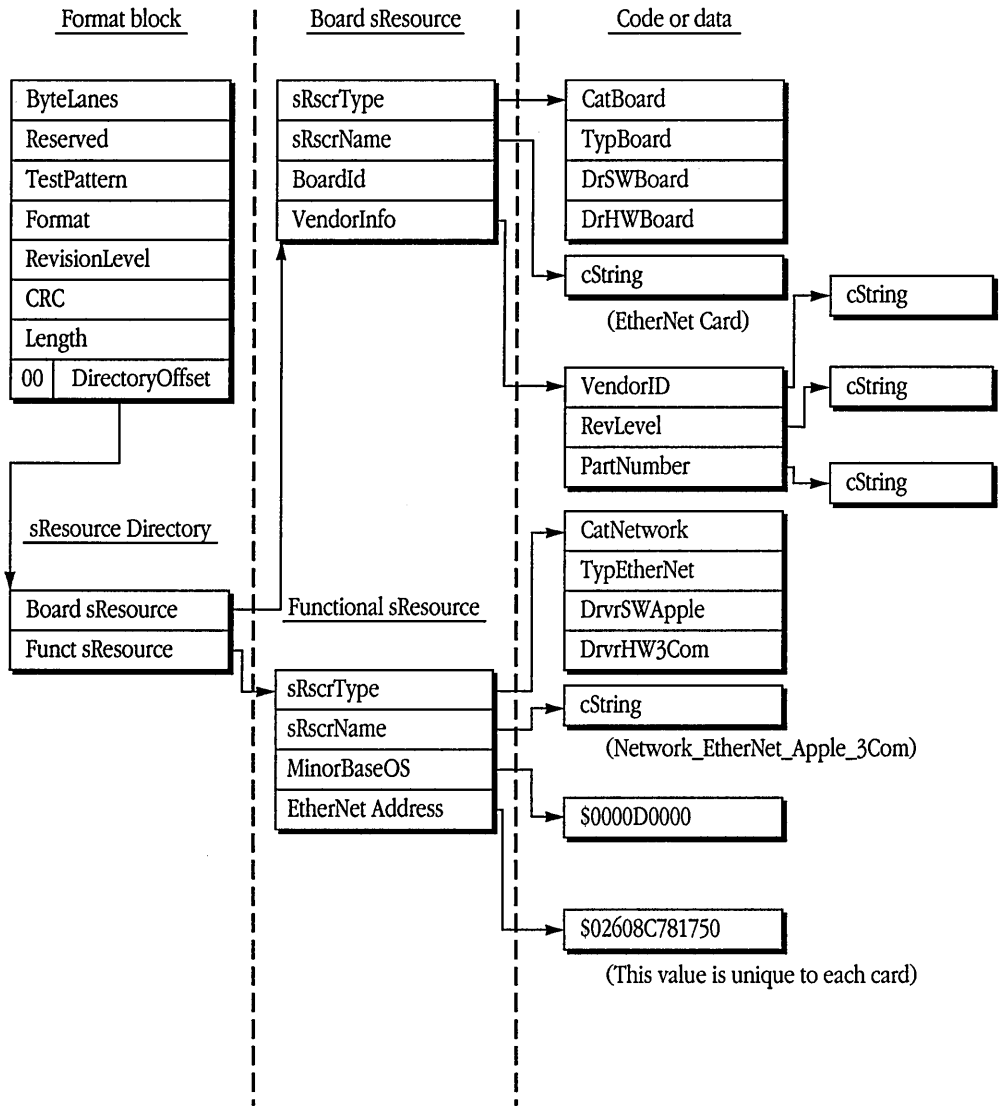
As a comparison, Figure 8-5 shows the simpler firmware structure of the Macintosh II EtherTalk™ Interface Card.

- *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



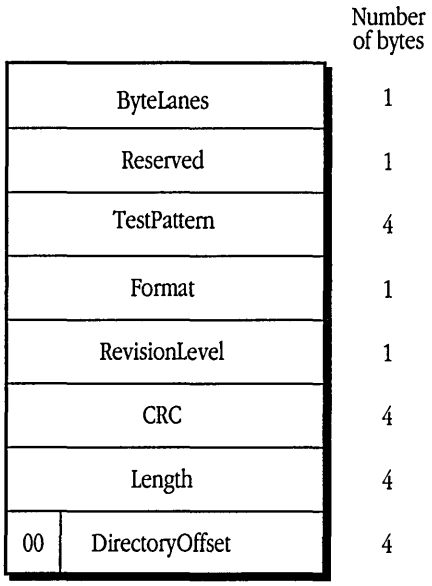
■ **Figure 8-5** Firmware structure of the Macintosh II EtherTalk Interface Card



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



The first byte of the format block must reside at one of the four 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 chapter of *Inside Macintosh.* For each slot it first searches NuBus addresses $FsFF FFFF–$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.

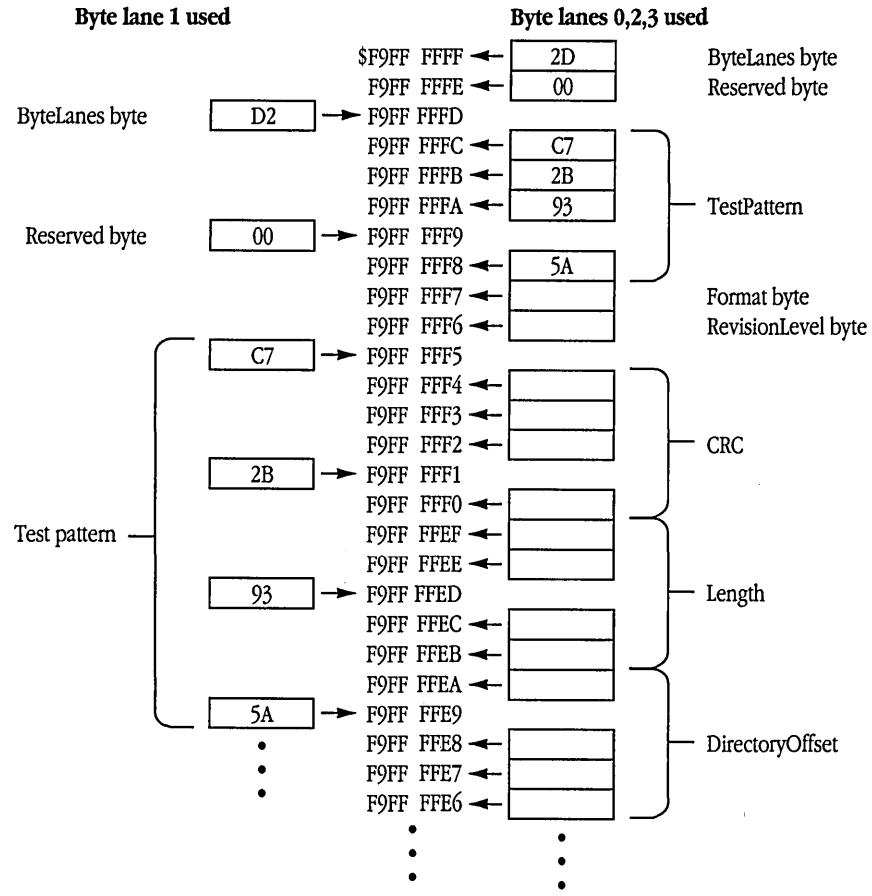
If no valid ByteLanes and TestPattern values are found, the Slot Manager stores a slot error in the corresponding sInfo record, as described in the Slot Manager chapter of *Inside Macintosh.*

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 byte lanes 0, 2, and 3 are used.

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

RevisionLevel

This one-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 in the form of a status value of –303.

CRC

The four-byte Cyclic Redundancy Check 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 one bit (with ROL.L #1 instruction)
        If pointer is pointing to the CRC field in the format header, goto @2
        Get the byte pointed to by pointer
        Add the byte to sum
@2      Increment pointer to next data byte
Goto @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 directory offset field in the declaration ROM image and the value of the ByteLanes in the ROM was $E1 (meaning every fourth byte was valid), then the DirectoryOffset would equal –$1000 even though the directory appears, to the central processor, to be $4000 bytes before the directory offset 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.

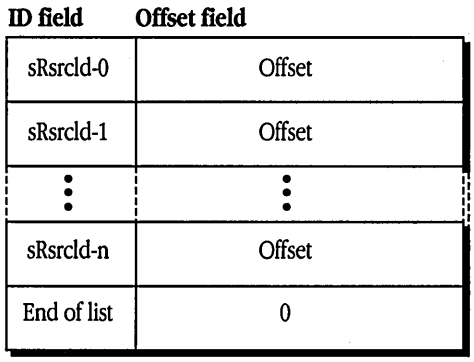
Each sResource defined by a card designer must have a unique identification number in the range 128–254. Identification number 255 is used as an end-of-list marker.

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



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

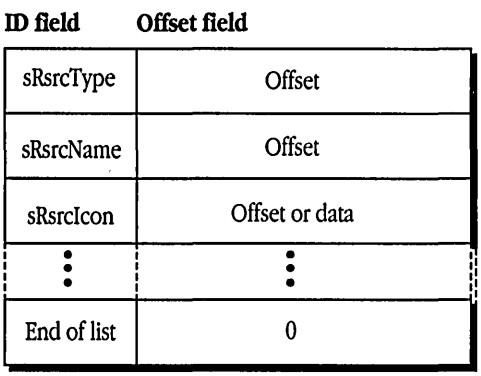
The last entry in the sResource directory must have an offset of 0 and an identification number of 255; that is, it must have the value $FF00 0000.

sResources

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



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 section "Sample Code," at the end of this chapter.

Apple-defined sResource entries

Table 8-3 lists 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

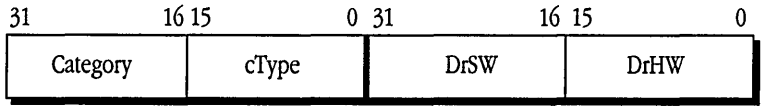
Name	ID number	Description
sRsrcType	1	Type of the sResource (required)
sRsrcName	2	Name of the sResource (required)
sRsrcIcon	3	Icon for the sResource
sRsrcDrvrDir	4	Driver directory for the sResource
sRsrcLoadRec	5	Load record for the sResource
sRsrcBootRec	6	Boot record
sRsrcFlags	7	sResource flags
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
sGammaDir	64	Gamma directory (for video cards only)

sRsrcType

The type entry in an sResource is used by the Macintosh Operating System or by an application or 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	cType 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 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 imbedded 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.

RsrcDrvrDir

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 'INIT' 31 resources 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.

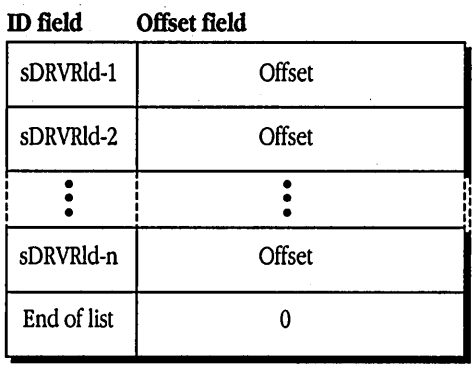
The actual value of the sRsrcDrvrDir entry in the sResource is an offset to an sDriver directory. Each entry in the sDriver directory contains an offset to an sDriver record and an sBlock containing the driver code. The identification number for each entry specifies which operating system supports the 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-family computer with MC68000 microprocessor
sMacOS68020	2	Driver will run on a Macintosh-family computer with MC68020 microprocessor
sMacOS68030	3	Driver will run on a Macintosh-family computer with MC68030 microprocessor

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

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



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 'INIT' 31 resources 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 the 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 chapter of *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 'INIT' resources have been executed, the sBootRec 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 parameter RAM)
→	seRsrcID	The sResource ID (from parameter RAM)
→	seDevice	The device number (from parameter RAM)
→	sePartition	The partition number (from parameter RAM)
→	seOSType	Type of operating system to boot (from parameter RAM)
→	seReserved	A reserved field (from parameter RAM)
→	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 parameter RAM)
→	seReserved	A reserved field (from parameter RAM)
→	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 in the dCtlDevBase field. 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 "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 x 32 x 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 x 32 x 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.

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 = spGammaDir = 64. For more information on the gamma table directory, see the section "Gamma Table Data," later in this chapter.

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.

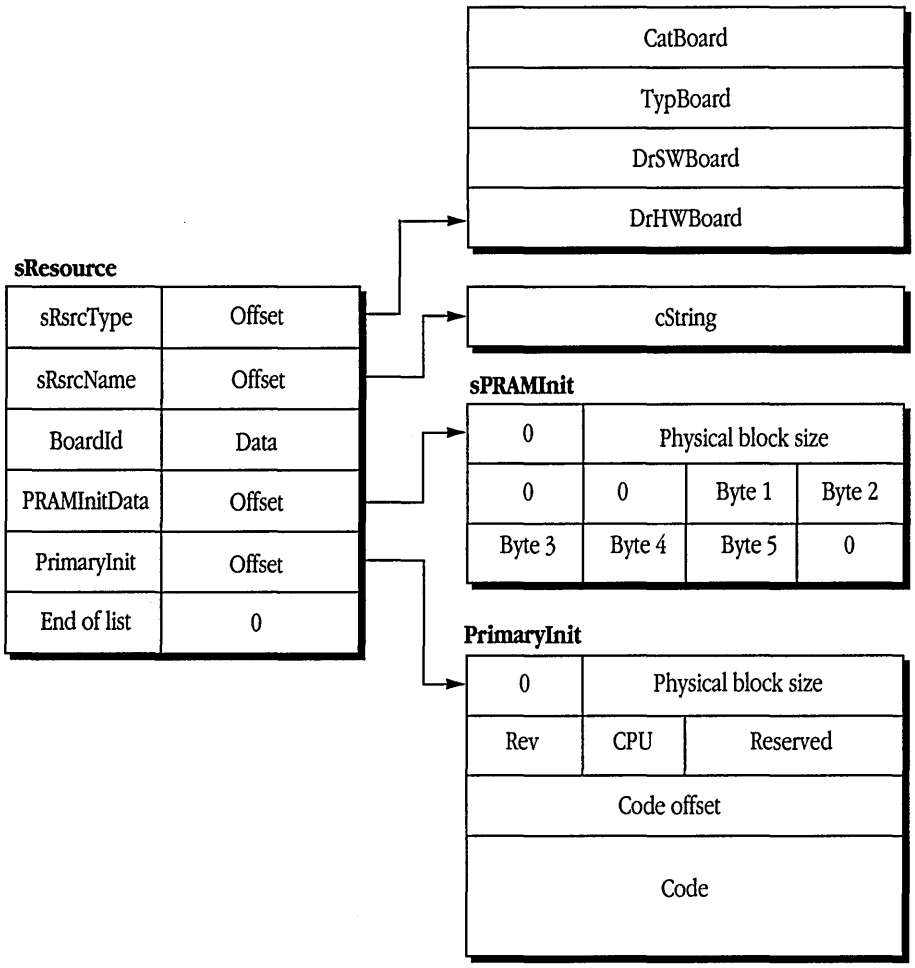
The entries in a Board sResource provide the computer with a card's identification number, vendor information, board flags, and initialization code. Table 8-5 lists the standard identification numbers assigned to the Apple-defined entries in the Board sResource. These entries are described in detail later in this section.

■ **Table 8-5** 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
sRsrcVidNames	65	Video mode name directory

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 8-12 shows the structure of a typical Board sResource.

■ **Figure 8-12** Typical Board sResource



BoardId

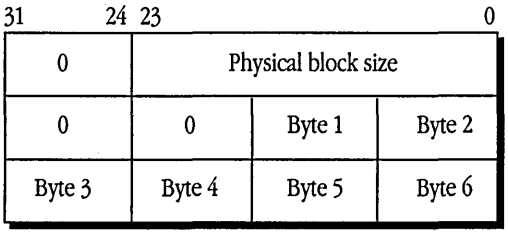
The BoardId is a required entry; without it, the computer will log an error in the appropriate sInfo record. The BoardId value is a word (two bytes) assigned by Apple. To obtain one for the card you are designing, contact Apple Macintosh Developer Technical Support.

PRAMInitData

There are six bytes reserved in the parameter RAM (PRAM) of a Macintosh II–family computer for each slot. The PRAMInitData entry lets you specify values other than zero for these bytes. If it is present in the Board sResource, the PRAMInitData 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 zero. Initialization occurs when the Macintosh Operating System detects a card for the first time or when the Slot Manager finds a BoardId in a Board sResource that is different from the BoardId in the corresponding sPRAMInit record.

The structure of the sPRAMInit record is shown in Figure 8-13.

■ **Figure 8-13** sPRAMInit record structure



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.
- 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 milliseconds.

Initialization code that exceeds these requirements can be placed in the Open routine of a driver provided for the card.

The 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 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.

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.

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 8-6.

■ **Table 8-6** 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 II–family 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 Color 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 SecondaryInits 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.

sRsrcVidNames

The optional sRsrcVidNames entry in a video card's Board sResource allows access to the video modes name directory. The video names directory identifies the various mode possibilities of video cards that operate in more than one video mode.

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 32-bit QuickDraw. The following sections describe these additions.

◆ *Note:* 32-bit QuickDraw is included in the CPU ROMs of the Macintosh IIci, the Macintosh IIfx, and any future computers in the Macintosh II family. You can also install it in earlier Macintosh II–family computers.

Identifying direct devices

The major focus of 32-bit 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 32-bit QuickDraw, Apple supported only **indexed video 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.

Setting the mVidType field in the video mode parameters to DirectType (2) allows 32-bit 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 gDevice's gdPMap descriptor.

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 the 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 should 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 should test the sRsrcFlags word for the device being written to.

If your Macintosh II–family computer does not have 32-bit QuickDraw in ROM, and if your card is intended as the startup device, make sure that you load 32-bit 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 Monitors cdev displays this icon in the Options dialog box for each different display. Monitors first searches the resource fork of the extension file for a 'cicn' = 0 resource, then examines the ROM for a color icon representing an equivalent resource image. If neither is found, Monitors searches for 'icl8' resources, followed by 'icl4' structures, and finally for 'ICON'. If no icon is found, Monitors displays a generic monitor icon. For more information on Monitors, see the chapter on the Control Panel in *Inside Macintosh*.

For a black-and-white icon, you can add an OSLstEntry with spID = sRsrcIcon = 3 to the Board sResource. This entry points to a standard 32 x 32 x 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 to the Board sResource. Color icons are in sBlock form because they are variably sized structures. 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.

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 dialog of the Monitors cdev 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 the Macintosh II Family," in Chapter 9, for a detailed description of gamma correction.

In each video sResource, an OSLstEntry with spID = spGammaDir = 64 points to a gamma directory. A list of OSLstEntries in the directory points to the various gamma table data structures. The gamma tables themselves are sBlocks—a length field followed by data. The spIDs of the gamma tables start at 128, the default gamma for this video mode, and increase by one 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 EndOfList entry.

The first field of each gamma block contains two 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,* Volume 5).

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 cdev gamma selector has a check box 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 cdev creates an additional option, Uncorrected, which 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, 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 mHRes and mVRes fields to the approximate characteristics of the monitor.

Sample code

Here is a sample of the declaration ROM firmware code for the Macintosh II Video Card using Macintosh Programmer's Workshop assembly language. Notice that this sample code reflects the configuration of the declaration ROM's firmware shown in Figure 8-4. Also included in this section are samples of the primary initialization code and the secondary initialization code for a video card.

```
;------------------------------------------------------------------------------
;
; (c) Apple Computer, Inc.   1989
; All rights reserved.
;
;------------------------------------------------------------------------------
;
;       File: SampleRom.a
;
;       This is a sample configuration ROM definition for a Macintosh II
;       Video Card. It demonstrates most of the features and capabilities
;       of video cards, including multiple monitor support, system code
;       identification, and manufacturer data structures.
;
;       This card supports the following modes:
;
;       1) 640*480 at 1-,4-,and 8-bits/pixel via 24-bit addressing
;       2) 640*480 at 1-,4-,8-, and 32-bits/pixel via 32-bit addressing
;       3) 640*870 at 1-,4-, and 8-bits/pixel
;       4) 640*480 at 1-bit per pixel only as a video mode family with 2 above
;
;------------------------------------------------------------------------------

;Include files
        PRINT   OFF

        INCLUDE         'SysEqu.a'              ;
        INCLUDE         'SysErr.a'              ;
        INCLUDE         'Traps.a'               ;
        INCLUDE         'VideoEqu.a'            ;
        INCLUDE         'ROMEqu.a'              ;
        INCLUDE         'QuickEqu.a'            ;
        INCLUDE         'SlotEqu.a'             ;
        INCLUDE         'DeclROMEqu.a'          ; generic declaration ROM EQU's
        INCLUDE         'SampleEqu.a'           ; equates specific to this ROM

        PRINT   ON


VideoDeclROM    MAIN
                STRING          C
                MACHINE         MC68020

;==============================================================================
;Constants
;==============================================================================

;
; sResource constants are in this form :  sRsrc_Vidxy,
;       where   x = size : S - 640*480, B - 640*870
;               y = addressing : 24 or 32
;               z = option : A - special mode
;
```

```
; These are the constants for the sRsrcs that are in the sRsrc directory. The numbering was
; developed in this fashion (to facilitate PrimaryInit):
;
;       Bit 7 = 1 always on for video sRsrc IDs
;       Bit 6 = 1 if optional 1-bit only mode, 0 otherwise
;       Bit 5 = 0
;       Bit 4 = 0
;       Bit 3 = 1 if big screen, 0 otherwise
;       Bit 2 = 0
;       Bit 1 = 0
;       Bit 0 = 1 if accessed in 32-bit mode, else 0 for 24-bit mode

sRsrc_Board    EQU     1               ; Board sResource
sRsrc_VidS24   EQU     $80             ;
sRsrc_VidS32   EQU     $81             ;
sRsrc_VidB24   EQU     $88             ;
sRsrc_VidS32A  EQU     $C1             ;

;=============================================================================
;       Directory (must be in ascending order!)
;=============================================================================

_sRsrcDir
        OSLstEntry      sRsrc_Board,_sRsrc_Board            ; Board sRsrc List
        OSLstEntry      sRsrc_VidS24,_sRsrc_VidS24          ; video sRsrc List
        OSLstEntry      sRsrc_VidS32,_sRsrc_VidS32          ; video sRsrc List
        OSLstEntry      sRsrc_VidB24,_sRsrc_VidB24          ; video sRsrc List
        OSLstEntry      sRsrc_VidS32A,_sRsrc_VidS32A        ; video sRsrc List
        DatLstEntry     EndOfList,0                         ;

;=============================================================================
;       sRsrc_Board List
;=============================================================================

_sRsrc_Board
        OSLstEntry      sRsrcType,_BoardType          ; offset to board descriptor
        OSLstEntry      sRsrcName,_BoardName          ; offset to name of board
        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                   ;

;
; Each NuBus 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            'Mac II Sample Video Card'    ; Name of Board

;
```

```
; The video name directory associates an optional name string with each video sRsrc present
; This name is read by the Monitors cdev and is presented in a video mode family selector in
; the Options dialog. If your card does not have mode families, it does not need a name
; directory. If you provide a monitor extension that allows family mode selection, once
; again, you may delete the name directory.
;

_sVidNameDir
        OSLstEntry      sRsrc_VidS32,_sNameReg       ; name record for regular mode
        OSLstEntry      sRsrc_VidS32A,_sNameOption   ; name record 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

;=============================================================================
;       Primary Init Record
;=============================================================================

_sPInitRec
        DC.L            _EndsPInitRec-_sPInitRec     ; Physical Block Size
        INCLUDE         'SamplePrimaryInit.a'        ; The Header/Code
        ALIGN   2
_EndsPInitRec

;=============================================================================
;       Secondary Init Record
;=============================================================================

_sSInitRec
        DC.L            _EndsSInitRec-_sSInitRec     ; Physical Block Size
        INCLUDE         'SampleSecondaryInit.a'      ; The Header/Code
        ALIGN   2
_EndsSInitRec

        STRING          C
;=============================================================================
;       Vendor Info record
;=============================================================================
;
; The vendor information record allows the developer to include revision level and part
; number information in their ROMs.
;

_VendorInfo
        OSLstEntry      VendorId,_VendorId    ; offset to vendor ID
        OSLstEntry      RevLevel,_RevLevel    ; offset to revision
        OSLstEntry      PartNum,_PartNum      ; offset to part number record
```

```
        DatLstEntry     EndOfList,0             ;
_VendorId
        DC.L            'Apple Computer'        ; Vendor Id
_RevLevel
        DC.L            'Sample 1.0'            ; Revision Level
_PartNum
        DC.L            'DAF-1234'              ; Part Number


;=============================================================================
;       sRsrc_Video
;=============================================================================

_sRsrc_VidS24
        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,_GammaDir1            ; directory for monitor type 1

; This is a list of the different modes supported by this hardware configuration

        OSLstEntry      FirstVidMode,_OBMs              ; offset to OneBitMode parms
        OSLstEntry      SecondVidMode,_FBMs             ; offset to FourBitMode parms
        OSLstEntry      ThirdVidMode,_EBMs              ; offset to EightBitMode parms

        DatLstEntry     EndOfList,0                     ;

;-----------------------------------------------------------------------------

_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                    ; this flag identifies 32-bit
                                                        ;   devices
        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,_GammaDir1            ;

;Parameters
        OSLstEntry      FirstVidMode,_OBMs              ; offset to OneBitMode parms
        OSLstEntry      SecondVidMode,_FBMs             ; offset to FourBitMode parms
        OSLstEntry      ThirdVidMode,_EBMs              ; offset to EightBitMode parms
        OSLstEntry      FourthVidMode,_DBMs             ; offset to Direct mode parms
        DatLstEntry     EndOfList,0                     ;

;-----------------------------------------------------------------------------

_sRsrc_VidB24
        OSLstEntry      sRsrcType,_VideoType            ;
        OSLstEntry      sRsrcName,_VideoName            ;
        OSLstEntry      sRsrc_DrvrDir,_VidDrvrDir       ;
```

```
        DatLstEntry     sRsrc_HWDevId,1                 ;

        OSLstEntry      MinorBaseOS,_MinorBase          ;
        OSLstEntry      MinorLength,_MinorLength        ;

        OSLstEntry      sGammaDir,_GammaDir2            ; directory for monitor type 2
;Parameters
        OSLstEntry      FirstVidMode,_OBMb              ; offset to OneBitMode parms
        OSLstEntry      SecondVidMode,_FBMb             ; offset to FourBitMode parms
        OSLstEntry      ThirdVidMode,_EBMb              ; offset to EightBitMode params

        DatLstEntry     EndOfList,0                     ;

;-------------------------------------------------------------------------------
;
; 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.
;
_sRsrc_VidS32A
        OSLstEntry      sRsrcType,_VideoType            ;
        OSLstEntry      sRsrcName,_VideoName            ;
        OSLstEntry      sRsrc_DrvrDir,_VidDrvrDir       ;
        DatLstEntry     sRsrcFlags,6                    ; for 32-bit devices
        DatLstEntry     sRsrc_HWDevId,1                 ;

        OSLstEntry      MinorBaseOS,_MinorBase          ;
        OSLstEntry      MinorLength,_MinorLength        ;

        OSLstEntry      sGammaDir,_GammaDir1            ;

;Parameters
        OSLstEntry      FirstVidMode,_OBMs              ; offset to OneBitMode parms

        DatLstEntry     EndOfList,0                     ;


;-------------------------------------------------------------------------------

_VideoType                                              ; Video sResource :
        DC.W            CatDisplay                      ; <Category>
        DC.W            TypVideo                        ;<Type>
        DC.W            DrSwApple                       ;<DrvrSw>
        DC.W            DrHwSample                      ;<DrvrHw>

_VideoName
        DC.L            'Display_Video_Apple_SampleCard'    ; this name must match the
                                                            ;   driver name

_MinorBase
        DC.L            defMinorBase                        ; frame buffer offset from
                                                            ;   dCtlDevBase
_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

        STRING          C

;=============================================================================
;       Gamma Directories
;=============================================================================

_GammaDir1                                            ; for the 640*480 display
        OSLstEntry      128,_StdGamma
        DatLstEntry     EndOfList,0

_GammaDir2                                            ; for the 640*870 display
        OSLstEntry      128,_BigGamma
        DatLstEntry     EndOfList,0

;
; Here's the image of the Apple HiRes RGB Monitor's standard gamma table.
;

_StdGamma
        DC.L            _EndStdGamma-_StdGamma

        DC.W            SGammaResID
        DC.B            'Mac II Std Gamma'
        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 ; gamma data
        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
```

```
_EndStdGamma

_BigGamma
        DC.L        _EndBigGamma-_BigGamma
        DC.W        BGammaResID
        DC.B        'Large Screen Gamma'
        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     ; gamma data
        DC.L        $2527282A,$2C2D2F30,$31333436,$37383A3B
        DC.L        $3C3E3F40,$42434445,$4748494A,$4B4D4E4F
        DC.L        $50515254,$55565758,$595A5B5C,$5E5F6061
        DC.L        $62636465,$66676869,$6A6B6C6D,$6E596F75
        DC.L        $20666F75,$6E642074,$68652073,$65637265
        DC.L        $74206D65,$73736167,$65212020,$47726565
        DC.L        $74696E67,$73206672,$6F6D2044,$61766964
        DC.L        $2046756E,$67A2A3A4,$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 for
                                                        ;   this configuration
        DatLstEntry     mPageCnt,Pages1s                ; number of video pages in this
                                                        ;   configuration
        DatLstEntry     mDevType,CLUTType               ; device type
        DatLstEntry     EndOfList,0                     ;

_OBMb
        OSLstEntry      mVidParams,_OBVParmb            ;
        DatLstEntry     mPageCnt,Pages1b                ;
        DatLstEntry     mDevType,CLUTType               ;
        DatLstEntry     EndOfList,0                     ;


_OBVParms
        DC.L            _EndOBVParms-_OBVParms          ; physical Block Size

        DC.L            defmBaseOffset
        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

_OBVParmb
        DC.L            _EndOBVParmb-_OBVParmb          ; physical Block Size

        DC.L            defmBaseOffset
        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


;
```

```
;=============================================================================
;       Four-Bit per pixel parameters
;=============================================================================

_FBMs
        OSLstEntry      mVidParams,_FBVParms            ;
        DatLstEntry     mPageCnt,Pages4s                ;
        DatLstEntry     mDevType,CLUTType               ;
        DatLstEntry     EndOfList,0                     ;

_FBMb
        OSLstEntry      mVidParams,_FBVParmb            ;
        DatLstEntry     mPageCnt,Pages4b                ;
        DatLstEntry     mDevType,CLUTType               ;
        DatLstEntry     EndOfList,0                     ;

_FBVParms
        DC.L            _EndFBVParms-_FBVParms          ; physical Block Size

        DC.L            defmBaseOffset
        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                  ;
        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            ;
        DatLstEntry     mPageCnt,Pages8s                ;
        DatLstEntry     mDevType,CLUTType               ;
        DatLstEntry     EndOfList,0                     ;

_EBMb
        OSLstEntry      mVidParams,_EBVParmb            ;
        DatLstEntry     mPageCnt,Pages8b                ;
        DatLstEntry     mDevType,CLUTType               ;
        DatLstEntry     EndOfList,0                     ;
EBVParms
        DC.L    _EndEBVParms-_EBVParms                  ; physical Block Size

        DC.L            defmBaseOffset
        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
        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 (32-Bit per pixel) parameters
;=============================================================================

_DBMs
        OSLstEntry      mVidParams,_DBVParms            ;
        DatLstEntry     mPageCnt,Pages32s               ;
        DatLstEntry     mDevType,DirectType             ; direct device type
```

```
        DatLstEntry     EndOfList,0                     ;

_DBVParms
        DC.L            _EndDBVParms-_DBVParms          ; 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            8                               ; bmPixelSize
        DC.W            3                               ; bmCmpCount
        DC.W            8                               ; bmCmpSize
        DC.L            defmPlaneBytes                  ; bmPlaneBytes
_EndDBVParms


SizeOf
        EQU     *

        WITH            FHeaderRec
        ORG             ROMSize-FHeaderRec.fhBlockSize
;=========================================================================
;       Format/Header Block
;=========================================================================
        DC.L            (_sRsrcDir-*)**$00FFFFFF        ;Offset to sResource directory
        DC.L            ROMSize                         ;Length of declaration data
        DC.L            0                               ;CRC {Patched by crcPatch}
        DC.B            AppleFormat                     ;Format
        DC.L            TestPattern                     ;Test pattern
        DC.B            0                               ;Reserved byte
        DC.B            $E1                             ;ByteLanes: 1111 0001


        ENDP

        END
```

```
;----------------------------------------------------------------------------------
;
; (c) Apple Computer, Inc.   1986-1989
; 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) Initialize the video frame buffer and video output
;              2) Disable VBL interrupts
;              3) Display a 50% dithered gray pattern on the screen
;              4) Perform any maintainence on Slot Manager structures
;
;       This sample card supports a number of configurations. These include
;       640*480 (addressed in 24- and 32-bit modes), 640*870, and, on machines
;       that have 32-bit QD in ROM, a 640*480 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    Begin-*                     ; offset to code

               WITH    seBlock,spBlock

Begin

;
; Set initial vendor status
;

                    MOVE.W  #seSuccess,seStatus(A0)     ; assume a good return

;
; Form 32-bit base address in A1
;

                    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.
;
;
; <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 desireable to identify
; the absence of a connected monitor, since it will allow CQD to remove this
; display from the desktop.
;
; In this example, we assume that the display type will return the type in D1.
; If the display is 640*480, D1=$80, if 640*870, D1=$88. These are the 24-bit
; flavored spID's for small and large display sRsrc lists. If no display is
; connected, this routine returns D1=$FF which is not a applicable spID. Later
; we will set the spID to disable in D7, so set it to $FF for now.
;

;
; <DEVICE-SPECIFIC>
; Initialize the video generation and CLUT here. You may init to any video mode,
; but it is preferable to initialize to 1-bit 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.
;

;
; 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)         ; 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
; 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 always have a 24-bit addressed video sRsrc, we simply match the monitor
; type if 32-bit QuickDraw isn't present. 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
; patch.
;
; The boot screen (the happy mac screen) is set to the depth in VendorUse1 before
; the System file is open. The other screens are set up per the scrn resource.
;
            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 re-setup 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  #$80,VendorUse1(SP)         ; select default depth

                CLR.B   scrnInval                   ; flag CQD that the scrn
                                                    ;   resource is bad
                MOVE.L  spResult(A0),spsPointer(A0) ; set up parameter block
                _SPutPRAMRec                        ; write the new record out

;
; 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
                ADDA    #SizesPRAMRec,SP            ; eliminate pram block

; 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_VidS32A,D4           ; is it the special mode?
                BEQ.S   @1                          ; yes
                MOVE.B  #sRsrc_VidS32,D4            ; set 32-bit mode as active
                MOVE.B  #sRsrc_VidS32A,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 Mac II Slot Manager is
; present, then this routine returns an error code in D0. If we don't have the new ; Slot Mgr
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.   1986-1989
; 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 was generally complete at
;       PrimaryInit.
;
;       Video Mode families depend upon inactivated video sResource lists
;       to be present. Since this is a new Slot Manager feature,
;       many 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 which 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
;

                    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 an 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)            ; external device = 0
                    CLR.L   spsPointer(A0)          ; we are adding resources that
                                                    ;   were in the sRsrc directory

;
; If the new Slot Manager was present in ROM, then activations and deactivation have
; already been performed. We also re-verify that 32-bit QuickDraw is around.
;
                    _sVersion                       ; find the Slot Manager version
                    CMP.L   #2,spResult(A0)         ; get the result (1= RAM patch,
                                                    ;   2=in ROM)
                    BEQ.S   SecInitDone             ; done
                    MOVE.L  #$A89F,D0               ; get the address of this trap
                    _GetTrapAddress ,NewTool        ;
                    MOVE.L  A0,D1                   ; save result
                    MOVE.L  #$AB03,D0               ; now test for new QD
                    _GetTrapAddress ,NewTool        ; get it too
                    CMPA.L  D1,A0                   ; are they the same?
                    BEQ.S   SecInitDone             ; if so, 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   spID(A0)                        ; start search at id=0
                    CLR.B   spTBMask(A0)                    ; we're going for an exact match
                    MOVE.W  #CatDisplay,spCategory(A0)      ; look for this card
                    MOVE.W  #TypVideo,spCType(A0)           ;
                    MOVE.W  #DrSwApple,spDrvrSW(A0)         ;
                    MOVE.W  #DrHwSample,spDrvrHW(A0)        ; look for our hardware
                    _sNextTypesRsrc                         ; get the spsPointer
                    MOVE.W  spRefNum(A0),D5                 ; save the refnum

;
; If the big screen sResource is installed, then we are done since it doesn't have
; a 32-bit addressed counterpart.
;
                    CMP.B   #sRsrc_VidB24,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 either be
; 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.W  spRefNum(A0)         ; just to be sure
                    CLR.L  spParamData(A0)      ; clear for activation
                    CLR.L  spsPointer(A0)       ; add back a 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)   ; assume that we are going to
                                                ; disable this mode
                    MOVE.B D4,spID(A0)          ; put in block
                    _sInsertSRTRec              ; insert and disable

;
; Test if we were the boot screen. If we were, 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
```

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 and gives you an example.

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 '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 'INIT' 31 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 chapter on 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.

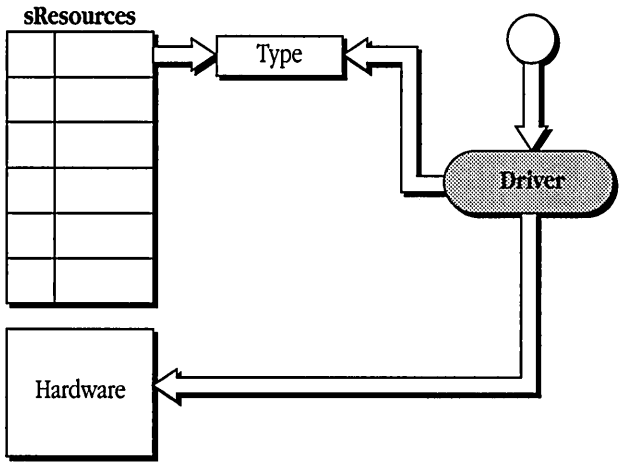
These two approaches are discussed in this section.

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

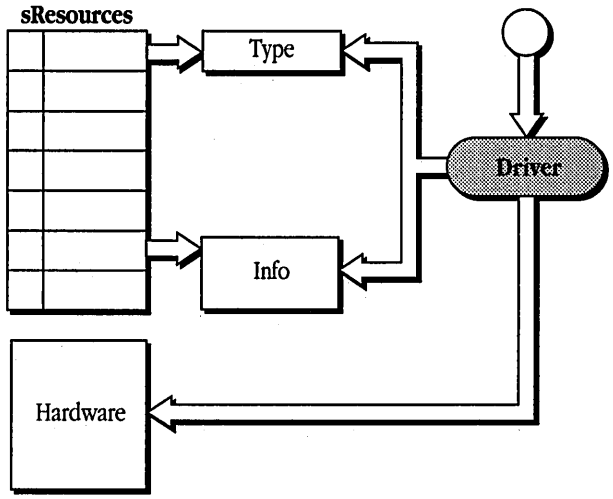


Card-generic drivers

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

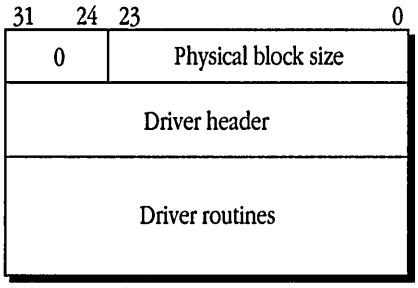


◆ *Note:* You can easily design a video-card declaration ROM that supports multiple video devices—for example, devices that work with different types of video monitors. At startup time, all sResources from the sResource directory are loaded into the slot device table. During PrimaryInit (and before any 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 under "The Structure of a Device Driver" in the Device Manager chapter of *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 the 'INIT' 31 resource to install them during startup. The 'INIT' 31 resource is described in the Start Manager chapter of *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 3–4. 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 an seBlock; on exit, the routine must return a handle in the seResult field of the same seBlock 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, or sMacOS68030, the system reads the driver from the card's ROM and installs it in the Macintosh system heap. These driver types are downward compatible; a 68030 CPU can execute sMacOS68020 and sMacOS68000 drivers as well as the sMacOS68030 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 'INIT' file to be added to the user's System Folder. The 'INIT' 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, 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.

◆ *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 II–family 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 longint 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 in the chapter on the Slot Manager 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 the 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.

```
; Device Control Entry Definition

DevCtlRecord   RECORD   0
dCtlDriver     DS.L     1              ; driver [handle]
dCtlFlags      DS.W     1              ; flags [word]
dCtlQueue      DS.W     1              ; queue header
dCtlQHead      DS.L     1              ; queue first-element [pointer]
dCtlQTail      DS.L     1              ; queue last-element [pointer]
dCtlPosition   DS.L     1              ; position [long]
dCtlStorage    DS.L     1              ; driver's private storage [handle]
dCtlRefNum     DS.W     1              ; refNum of this driver [word]
dCtlCurTicks   DS.L     1              ; counter for timing systemTask calls
                                       ; [long]
dCtlWindow     DS.L     1              ; driver's window (if any) [pointer]
dCtlDelay      DS.W     1              ; number of ticks between systemTask calls
```

```
                                        ; [word]
dCtlMask        DS.W    1               ; desk accessory event mask [word]
dCtlMenu        DS.W    1               ; memu Id associated with driver
[word]
dCtlSlot        DS.B    1               ; device slot number [byte]
dCtlSlotIdq     DS.B    1               ; device Id within slot [byte]
dCtlDevBase     DS.L    1               ; driver scratch ptr/offset from base
to                                      ; device [long]
dCtlOwner       DS.L    1               ; ptr to task control block
(ownership)                             ; [Ptr]
dCtlExtDev      DS.B    1               ; Id of external device [byte]
                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 the hardware base address (from dCtlDevBase) in this field.

The Device Manager in the Macintosh II family of 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 $9 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 ($9–$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 nonzero to indicate 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 either 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 one 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 eight bytes of dedicated parameter RAM (PRAM). The first two bytes cannot be modified and always contain the card's Board ID. Normally, the other six 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 declaration ROM spIDs

Name	ID number	Description
mVidParams	1	Video device record ID
mTable	2	Offset to the device color table for fixed CLUT devices
mPageCnt as a	3	Number of video display pages for this mode (expressed counting number)
mDevType	4	Device type (0 = indexed CLUT device, 1 = indexed fixed device, and 2 = direct device)

The declaration ROM for aX 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 = chunky indexed; $10 = chunky direct)
vpPixelSize	Word	Number of bits in pixel (rounded upward to the next power of 2 for chunky indexed and chunky direct 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 chapter on the Device Manager in *Inside Macintosh.* This section discusses only requirements specific to video drivers.

Normally, a driver associated with a Macintosh II–family 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 one 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

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 as determined by parameter RAM 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 during reboots

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
VDEntRecPtr = ^VDEntryRecord;
VDEntryRecord = RECORD
                  csTable: Ptr;        {pointer to color look-up table}
                  csStart: INTEGER;    {start entry number}
                  csCount: INTEGER;    {count number}
                END;
VDGamRecPtr = ^VDGammaRecord;
VDGammaRecord = RECORD
                  csGTable: ptr;       {pointer to gamma table (see graphics devices
                                              chapter in Inside Macintosh)}
                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;
              pad: signedByte;
            END;
VDDefModePtr = ^VDDefModeRec;
VDDefModeRec = RECORD
                 spID: signedByte;
                 pad: signedByte;
               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 driver status call were returned directly in the I/O parameter block. Using the new convention, results from the driver status call are returned directly to csInfoBlock.

Slot information applicable to the card associated with your video driver is contained in the device control entry, as described in the Device Manager chapter of *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 will see references to video devices that operate in indexed pixel mode (commonly called *indexed devices*) and devices that operate in direct pixel mode (commonly called *direct devices*). The section "Additional Firmware Requirements of Video Cards" in Chapter 8 describes indexed and direct 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 128.

csCode = 0	**csParam**	= **VDPgInfoPtr**	**[Init]**
←	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 one 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 initialize 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 parameter RAM.

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 the csTable to determine which entries are to be changed. Both csStart and csCount are zero-based; their values are one less than the desired amount. For a description of the structure of a color look-up table, refer to the information on Color QuickDraw in *Inside Macintosh.*

◆ *Note:* The csStart value refers to logical position, not physical position. In four-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- and 32-bit video modes associated with direct devices, the display color is implied directly by the pixel value, and there is no color table or color matching in effect. 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.

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 in the Graphics Devices chapter of *Inside Macintosh*. Gamma correction is defined and explained later in this chapter in the section "Gamma Correction in the Macintosh II Family."

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 over-saturated 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, 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, 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 color look-up table 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 of the new choice's sResource in the card's slot parameter RAM. The Monitors cdev 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 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 saved in slot PRAM is compatible with the current environment, and if so, 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 five 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.

csCodes 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 color look-up table 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 the csTable to determine which entries are to be returned. Both csStart and csCount are zero-based; their values are one 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.

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 color look-up table 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 pre-allocate 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.

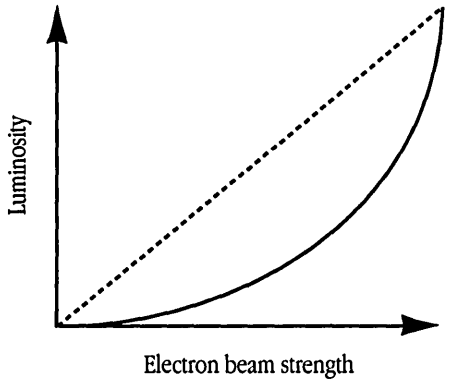
Gamma correction in the Macintosh II family

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. The Macintosh II family of 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

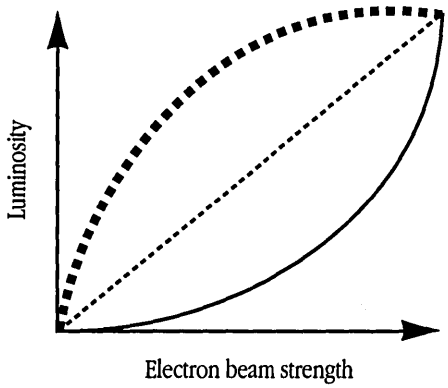
As the beam from a video display's electron gun sweeps the scan lines, it strikes phosphors on the face of the monitor tube and causes them to luminesce. If you increase the intensity of the beam, the phosphor dots luminesce more brightly, and if you reduce the intensity of the beam, the phosphor dots 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 non-linear 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 eight bits of each channel to reference the gamma table.

The gammaTbl data structure

Following is the structure of the data table 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                 {data for gamma calculation
                      [0.. gFormula size]    formula}
                      of byte;
    gData           : array [0..gData Cnt]  {gamma correction look-up
                      of byte;               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 that this table was measured for. 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 the Macintosh II–family 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 in the chapter on the Resource Manager 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

Here is an example of a possible video card driver, written in Macintosh Programmer's Workshop assembly language.

```
;-----------------------------------------------------------------------------
;
; (c) Apple Computer, Inc. 1989
; All rights reserved.
;
;-----------------------------------------------------------------------------
;
;       File   : SampleDrvr.a
;
;       This file contains a sample video driver for use by the Macintosh
;       OS in MPW 3.0 format. It is structured as a normal Mac slot device driver.
;       It is assembled as part of the declaration ROM image, so this file does
;       not need 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>.
;
;       The theoretical video card supports 1-,4-,8-, and 32-bit/pixel depths
;       on 640*480 and 1-,4- and 8-bit on 640*870 displays. Also, in systems that
;       have the version 2.0 Slot Manager and 32-bit QuickDraw, there is a 1-bit
;       only version of the 640*480 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 which 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 element
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 patch
; inits. 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_SampleCard    ; video driver name
                STRING          ASIS
                ALIGN           2                           ; make sure we're 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.
;
; Remember that all state information should be kept in private storage to allow the driver
; to be shared between multiple identical cards in a machine.
;
; 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 private
                                                        ;   storage
                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 this card
            MOVE.W      #TypVideo,spCType(A0)
            MOVE.W      #DrSwApple,spDrvrSW(A0)
            MOVE.W      #DrHwSample,spDrvrHW(A0)    ; look for our hardware
            _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 640*870 display is connected then bit
; 3 is set, and if we are in the special one-bit only 640*480 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 640*480
                                            ;   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 640*870
                                            ;   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 (always 128)
            _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           #0,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 relative offset to the
                                ;   routine
                JMP             CtlJumpTbl(PC,D0.W)         ; go to the proper routine

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 FirstVidMode as requested
                                                            ;   mode
                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 (clearing
                                                            ;   DirectFlag)
                BSR             HWSetPage                   ; set the page from D0
                MOVE.L          D0,saveBaseAddr(A3)         ; save the new base address in
                                                            ;   private storage
                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           #FourthVidMode,csMode(A2)   ; is this 32-bit/pixel mode?
                BEQ.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 which 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 (32-bits/pixel), then this routine
; returns an error.
;
; This code is shared with DirectSetEntries, below. Since luminance mapping should not occur
; in direct modes, the code which 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
;          non-zero, it should write immediately to the CLUT hardware.
;

        WITH VidLocals
                MOVE.L          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.
;
; 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 the 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.
;
;       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, then accept a generic 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 one
                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                 ; set the 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
                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                          ; all done

        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, just in case 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             ; set the RGB channels up 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
;   grayscale 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.B          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                 ; setup 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)     ; setup queue ID
                LEA             BeginIH,A2              ; get pointer to interrupt handler
                MOVE.L          A2,SQAddr(A0)           ; setup int routine address

                MOVE.L          A1,SQParm(A0)           ; this field is passed to the interrupt
                                                        ;       handler, and can be any convenient
                                                        ;       value (in this case the DCE handle)

                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 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.
;       Non-compatible 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 a slot parameter block 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 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 setup 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.L      A3,-(SP)              ; save A3
            MOVE.L      dCtlStorage(A1),A3
            MOVE.L      (A3),A3               ; get pointer to private storage

            BSR         DisableVGuts          ; call utility to deactivate interrupts

            MOVE.L      saveSQElPtr(A3),A0    ; get interrupt handler queue element
            _DisposPtr                        ;   dispose it
            MOVE.L      saveGammaPtr(A3),A0   ; get pointer to gamma table
            _DisposPtr                        ;   and 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.
                LSL.W           #1,D0                       ; Adjust csCode to be an index into
                                                            ;   the table.
                MOVE.W          StatJumpTbl(PC,D0.W),D0     ; Get the relative offset to the
                                ;   routine.
                JMP             StatJumpTbl(PC,D0.W)        ; go to the proper routine.

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.L          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.
;

                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 zero
                                                            ;   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 0 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.L          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 0 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
                CMP.W           #FourthVidMode,D1           ; is it the direct video mode?
                BEQ.S           Fill32                      ; different code for big pixels
                SUB             #OneBitMode,D1              ; make it 0 based
                MOVE.W          D1,D3                       ; make a copy of it

                LEA             Pats,A1                     ; get a pointer to the pattern table
                MOVE.L          (A1,D3*4),D5                ; D5 = the proper pattern
                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             (ZA0,D0)                    ; OK with the 68020
@Continue32
                MOVEQ           #true32b,D0                 ; switch to 32-bit addressing mode
                _SwapMMUMode                                ; flip to 32
                MOVE.L          D0,-(SP)                    ; save previous MMUMode
NxtRow1         MOVE.L D4,D2                                ; load it into count index
NxtLong1        MOVE.L          D5,(A0)+                    ; write long word pattern to screen
                DBF             D2,NxtLong1                 ; for the entire line
                NOT.L           D5                          ; invert pattern
                DBF             D3,NxtRow1                  ; do it for all screen lines
                MOVE.L          (SP)+,D0                    ; restore MMU mode
                _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
```

```
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, 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
;                4-bit = $81
;                8-bit = $82
;               32-bit = $83
;
;----------------------------------------------------------------------------

Small24Parms
MaxDepth        DC.W            $82                     ; 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            Pages4s,RB4s            ; pages, rowbytes
                DC.W            Pages8s,RB8s            ; pages, rowbytes

Small32Parms
                DC.W            $83                     ; maximum screen depth ID
                DC.W            defmBounds_Bs           ; screen height
                DC.W            Pages1s,RB1s            ; page and rowbyte info
                DC.W            Pages4s,RB4s
                DC.W            Pages8s,RB8s
                DC.W            Pages32s,RB32s

BigParms
                DC.W            $82                     ; maximum screen depth ID
                DC.W            defmBounds_Bb           ; screen height
                DC.W            Pages1b,RB1b            ; page and rowbyte info
                DC.W            Pages4b,RB4b
                DC.W            Pages8b,RB8b

OneParms
                DC.W            $80                     ; maximum screen depth ID
                DC.W            defmBounds_Bs           ; screen height
                DC.W            Pages1s,RB1s            ; page and rowbyte info

D_MaxDepthID    EQU             MaxDepth-Small24Parms
D_Height        EQU             Height-Small24Parms
D_Pages         EQU             Pages-Small24Parms
D_RowBytes      EQU             RowBites-Small24Parms
```

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
VDEntRecPtr = ^VDEntryRecord;
VDEntryRecord = RECORD
                  csTable: Ptr;              {pointer to color look-up table}
            csStart: INTEGER;                {start entry number}
            csCount: INTEGER;                {count number}
                END;
VDGamRecPtr = ^VDGammaRecord
VDGammaRecord = RECORD
            csGTable: ptr;           {pointer to gamma table (see graphics devices
                                chapter in Inside Macintosh}
                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;
```

Interrupt queue routines

```
FUNCTION SIntInstall(sIntQElemPtr: SQElemPtr; theSlot: INTEGER): OsErr;
FUNCTION SIntRemove(sIntQElemPtr: SQElemPtr; theSlot: INTEGER): OsErr;
```

Advanced control routines

csCode	csParam	Effect
0	VDPgInfoPtr	Resets card to startup state
1	VDPgInfoPtr	Stops and purges I/O requests
2	VDPgInfoPtr	Changes card's video mode
3	VDEntRecPtr	Changes card's color table (if any)
4	VDGamRecPtr	Creates a gamma table
5	VDPgInfoPtr	Fills video page with gray
6	VDPgInfoPtr	Selects actual colors or luminance-equivalent gray tones
7	VDFlagPtr	Controls VBL interrupts
8	VDEntRecPtr	Changes color table (direct device only)
9	VDDefModePtr	Sets default configuration in PRAM

Advanced status routines

csCode	csParam	Effect
0 and 1		Not implemented in video drivers
2	VDPgInfoPtr	Returns card's current video mode, page, and base address
3	VDEntRecPtr	Returns color table entries
4	VDPgInfoPtr	Returns number of video pages available
5	VDPgInfoPtr	Returns base address of specified page
6	VDFlagPtr	Returns selection status of actual colors or luminance-equivalent gray tones
7	VDFlagPtr	Indicates status of VBL interrupts
8	VDGamRecPtr	Returns a pointer to current gamma table
9	VDDefModePtr	Returns default value of video sResource's spID entry

Assembly-language information

Data structures

```
;Use with control and status calls where csCode = 3
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
```

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 for the Apple implementation of the NuBus interface in the Macintosh II family.

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 II–family 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 Table 3-1, "Transfer Mode Coding"). 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 eight 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
$Fss8 0000	Master (write-only) register
$Fss0 0000	Address register
$Fss4 0000	Data 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 1-4 and Figure 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 would 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 5 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. For example, 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 II family, it is important to make a clear distinction of whether a value is specified as viewed from the perspective of the computer's microprocessor or 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 complement 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 two bits of the Address register become part of the NuBus transfer mode; the values of these two 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.

Examples

This section describes 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 (0/1). 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 of $0000 1234 (Macintosh II–family 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 (of $5678) to $F900 1234 (slot 9) after $40 clock cycles: 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 two 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 the $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 5 at the end of the book. PAL equations are displayed in Appendix B.

The NTC consists of

- four NuBus address/data buffers (74ALS651s), U1–U4 (also called transceivers)
- eight octal latches (74ALS374s), U5–U12, which implement the Address and Data registers
- one ROM socket, U13, for the declaration ROM
- two 4-bit counters (74ALS161s), U23–U24, which implement the Delay counter
- one 74F86, U14, and one 74F30, U15, which form an address comparator
- five PALs, U16–U20, which implement the control logic
- 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 upon either 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, 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.

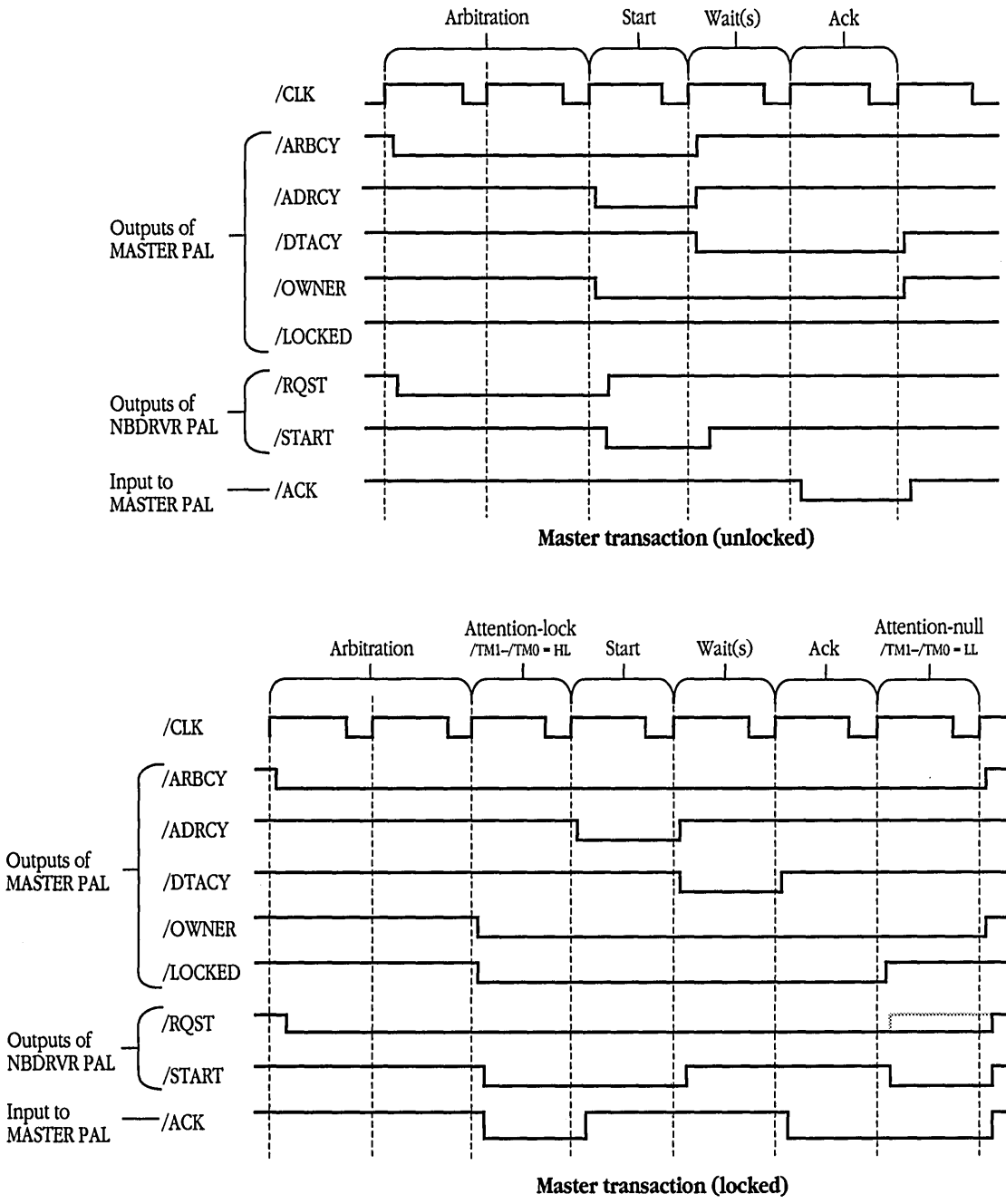
MISC PAL

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



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, it 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. /RQST 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

This card is an example of how a simple, 8-bit I/O chip may be supported over NuBus.

The SCSI-NuBus Test 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.

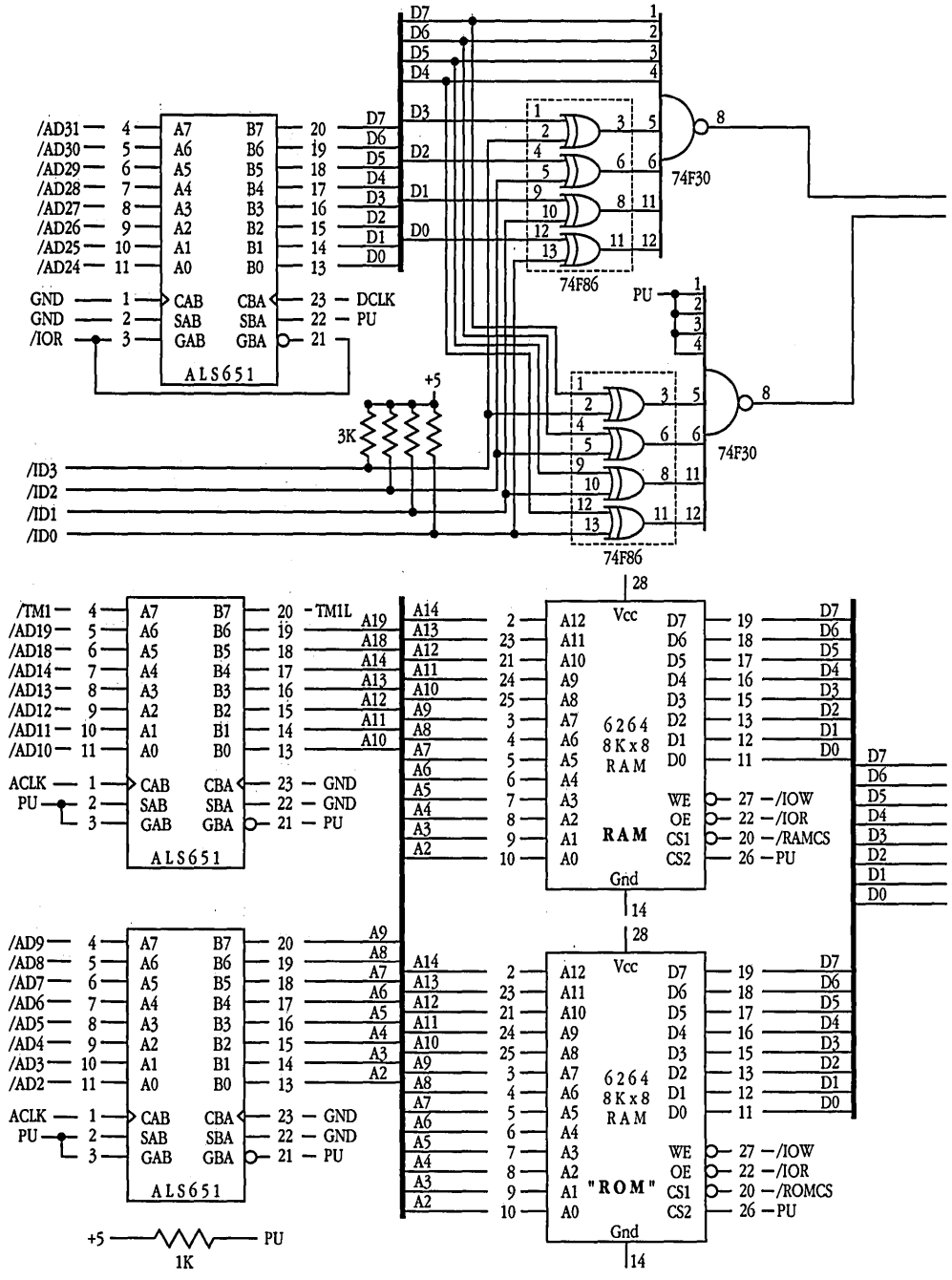
The SCSI chip can generate NuBus interrupts (via /NMRQ) from both IRQ and DRQ; this interrupt can be disabled.

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 two 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

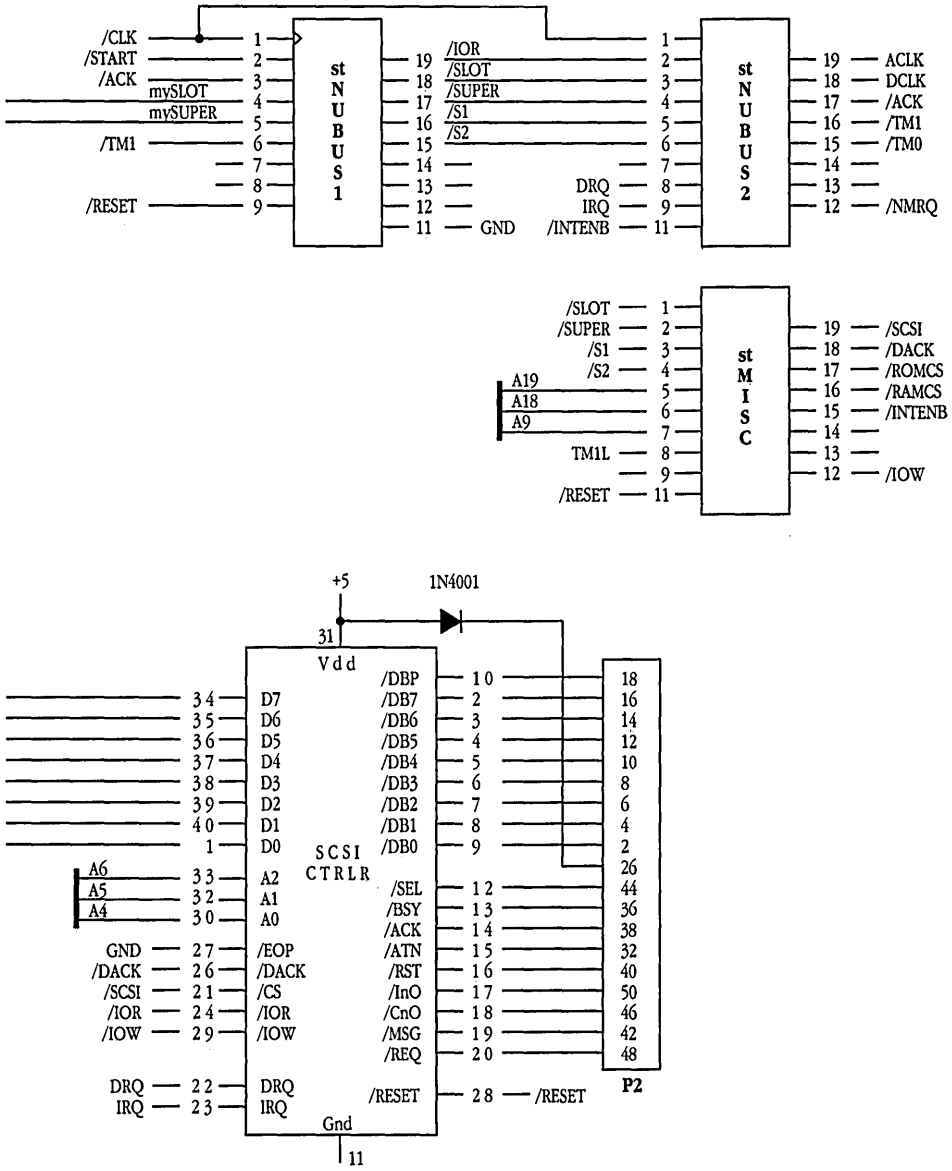
This section describes the hardware components and how they function. Figure 10-2 is an electrical schematic of the SCSI-NuBus Test Card; Figure 10-3 is the timing diagram. The PAL equations are in Appendix C.

■ **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.

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 byte-wide 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)

This 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)

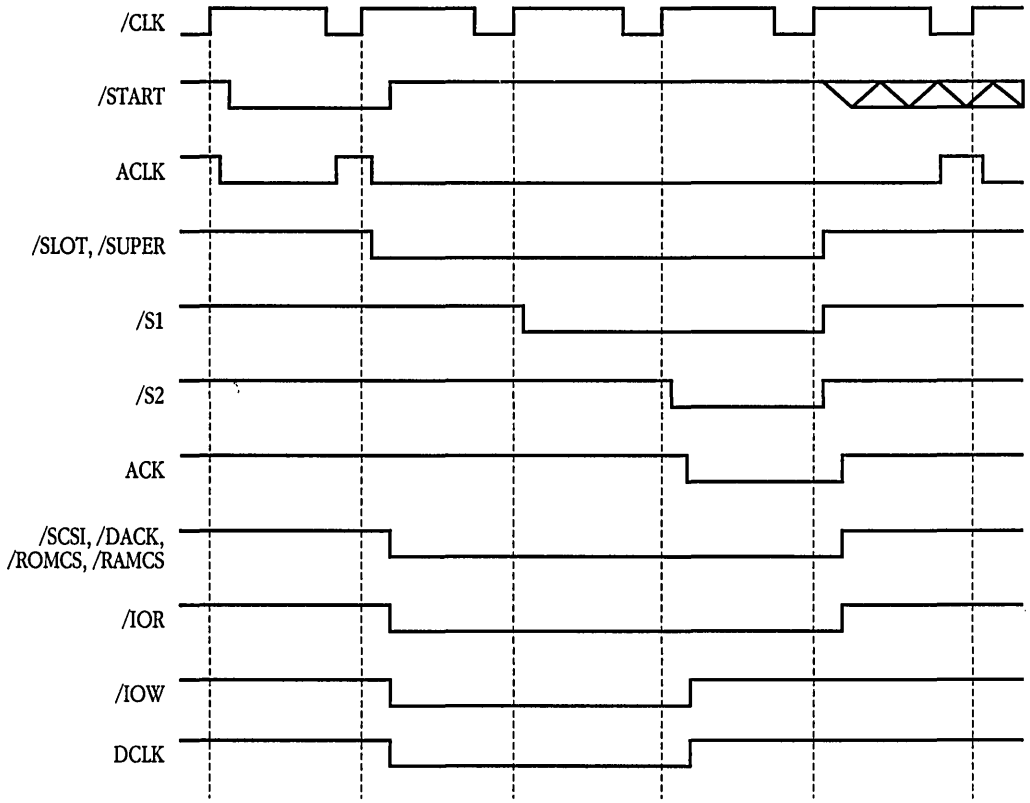
This PAL (16R8B) performs the basic NuBus timing for the card. When either mySLOT or mySUPER is detected during a start cycle, the PAL generates /SLOT or /SUPER and starts a 2-bit counter (/S2, /S1), which is used by s/TMISC. The value of /TM1 during the start cycle is latched to form the /IOR signal, the assertion of which indicates a read.

NuBus signal generator (stNUBUS2 PAL)

This PAL (16L8B) decodes the state of /SLOT, /SUPER, and /S2 to generate the acknowledge cycle and control the latching of the 651's.

stNUBUS2 is also used to generate the open-collector /NMRQ signal for presentation of interrupts to the main logic board.

■ **Figure 10-3** SCSI-NuBus timing diagram



Decode and timing (stMISC PAL)

This 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)

This 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 x 8 RAM which can be written using the ROM address space. However, a real 8 KB x 8 ROM may be inserted instead.

RAM

The RAM chip is an 8 KB x 8 RAM which is accessible only by addressing super slot space.

PAL descriptions

The source code of the three PALs is in Appendix C. Refer to these listings, along with the timing diagram and schematic, 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 II–family 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: These 74LS640-1's buffer the internal address/data bus of the controller from the NuBus address/data bus.

Address counters: These 74LS169 counters 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: This is the 2048 x 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: This 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.

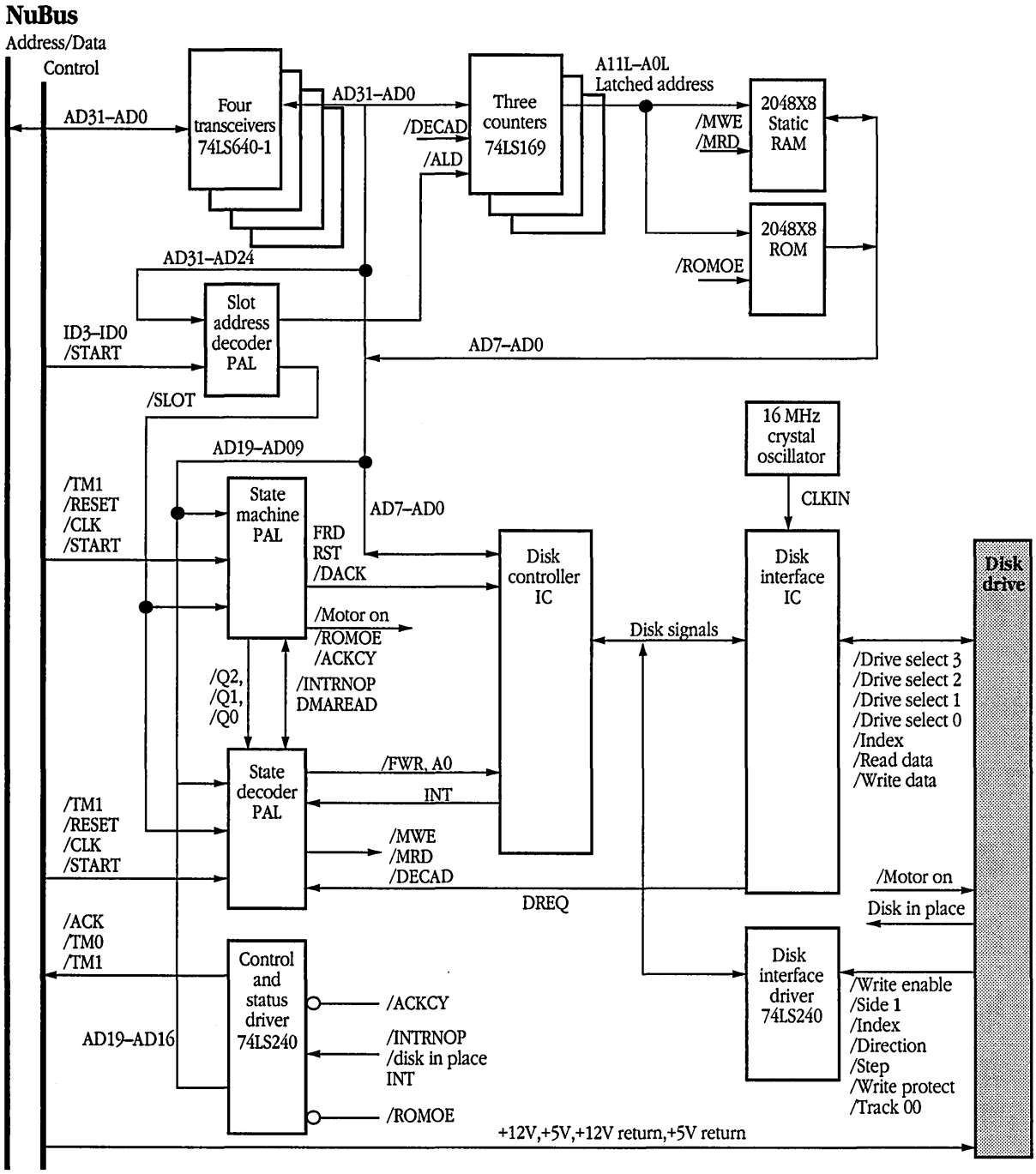
Slot address decoder PAL: This PAL20L10 determines if the controller's slot address is selected. It uses the signal /START and address decoding to compare if the upper nibble of the address is an $F and if the address lines A27–A24 and D3–D0 compare with the hard-wired slot ID address.

State machine PAL: This PAL20X10 generates the timing for programmed I/O and internal DMA transfers on the controller.

State decoder PAL: The state number is decoded by this 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: This 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: This 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: This 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 they 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 which 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 signals are described in Table 10-3.

■ **Table 10-3** RAM access signals

Signal name	Signal description
/SLOT	Signals that a NuBus cycle to the controller is active
/ALD	Used to load the RAM/ROM address into the address counters; gates the clock signal into the synchronous counters
/FWR	Enables disk controller IC write enable
/FRD	Enables disk controller IC read enable
A0	Disk controller IC register select: 0 selects main status register, 1 selects data register
SR	Direction signal to bidirectional driver on the address/data bus: 0 means write to NuBus, 1 means read from NuBus
/DREQ	Requests DMA cycle from disk controller IC or disk interface IC
/DACK	Acknowledges the DMA cycle requested
/ACKCY	Gates /ACK and /TM1–/TM0
/MWE	Enables RAM memory write
/MRD	Enables RAM memory read output
/INTRNOP	When asserted, indicates internal DMA operation in process
/DMAREAD	Indicates a DMA read operation when asserted
/DECAD	Enables the DMA address counters to decrement by one memory location

Programmed I/O (PIO) operations

Control and status information is passed to and from the controller using programmed I/O operations. PIO transfers include RAM and disk controller IC reads and writes, and ROM reads. The Motor On 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 decoder PAL. It asserts the signals /SLOT and /ALD. /SLOT indicates that the NuBus cycle is currently active. /ALD is used as a clock enable signal for loading the RAM or ROM address into the counters. /ALD 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

Transfers to and from the sector buffer RAM are done by on-card DMA. DMA is 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 zero 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 up 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 II 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 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 from 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.

In order 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. This purpose of this information is to provide you with an overview of good video card design, but not 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.

Video card overview

The original Macintosh II Video Card and the new 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 x 480 pixels
- refresh rate of 67 hertz for reduced flicker
- up to 256 colors out of 16 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 newer 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.*

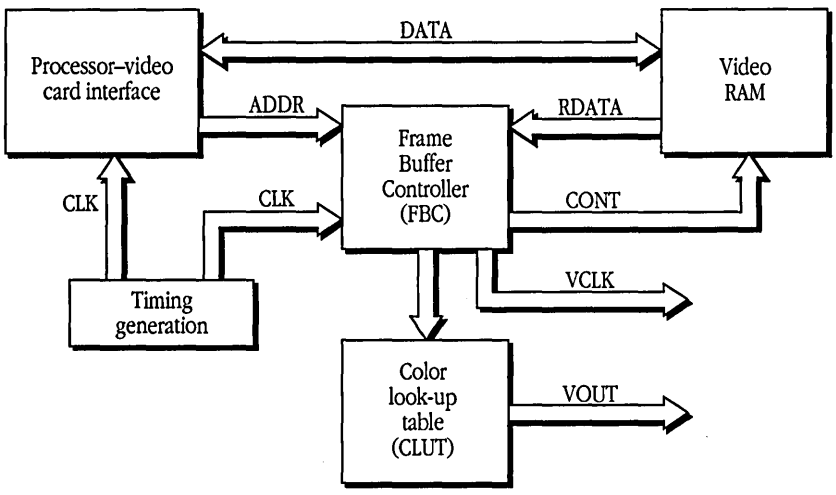
△ **Important** In addition to the 256 KB to 512 KB of video memory that QuickDraw manages, most of the video card features are subject to software control through several control addresses. These addresses are all located in the 16 MB slot space described in Chapter 7, "NuBus Card Memory Access." Since there is a difference between NuBus address allocation and the mapping of address space in the Macintosh II family, 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



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 (FBC)

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 x 4 device with 150 nanosecond 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 percent 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:

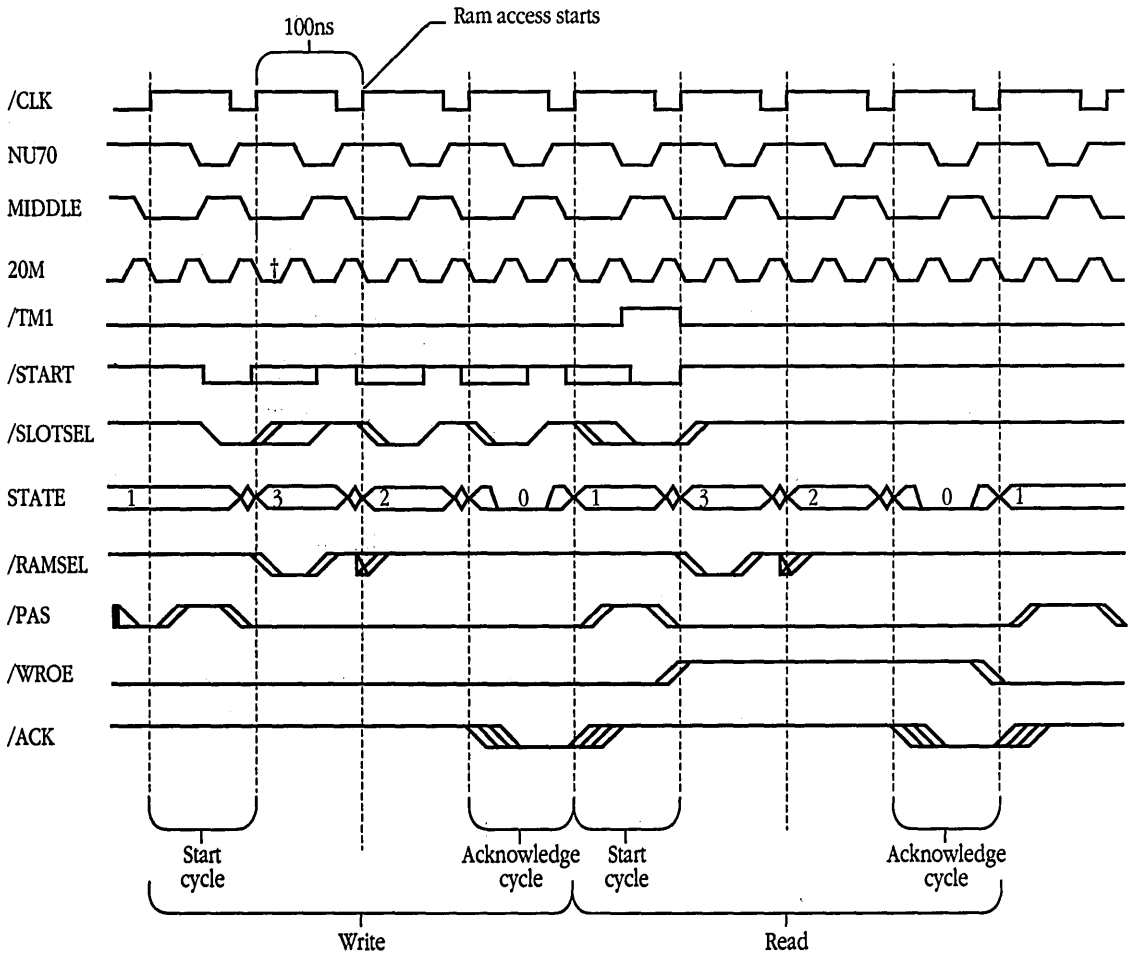
1. The current bus master drives /START to asserted (low), places desired video RAM space *address* on the /AD31–/AD2 bus, and drives </TM1–/TM0,/AD1–/AD0> with the transfer mode. /TM1 is low when the first transaction in Figure 11-2 starts, indicating that a write transaction is under way.
2. Write 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 (CLUT)

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.

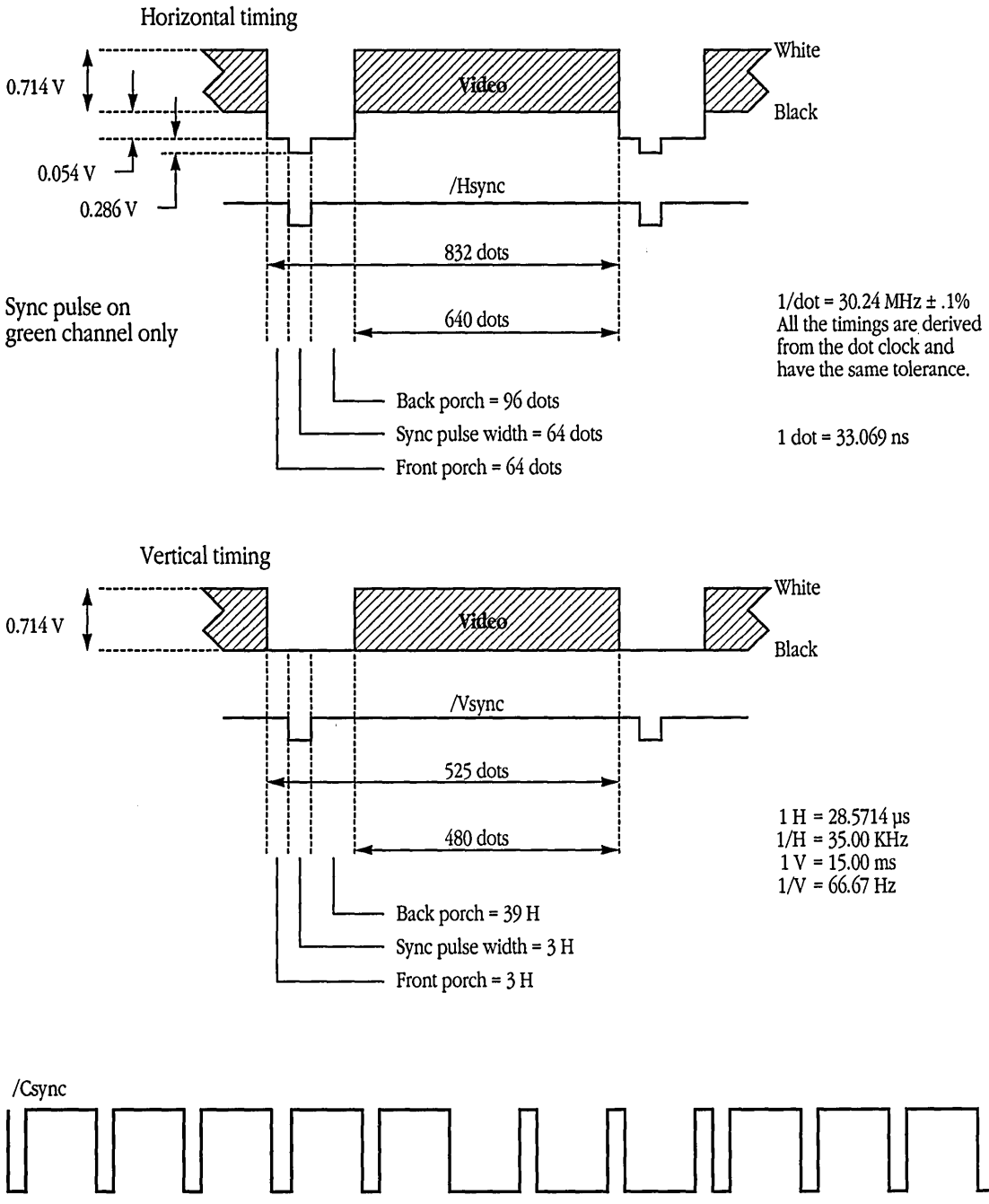
The CLUT is the electrical interface between the FBC and the analog video output device. In operation, the FBC controls the transfer of digital pixel data from the video RAM to the CLUT. Inside the CLUT is a table of RGB triples, one for each currently available color. Each pixel's worth (1, 2, 4, or 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 eight 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 or RS-170–compatible RGB video signals to the video connector at the rear of the video card.

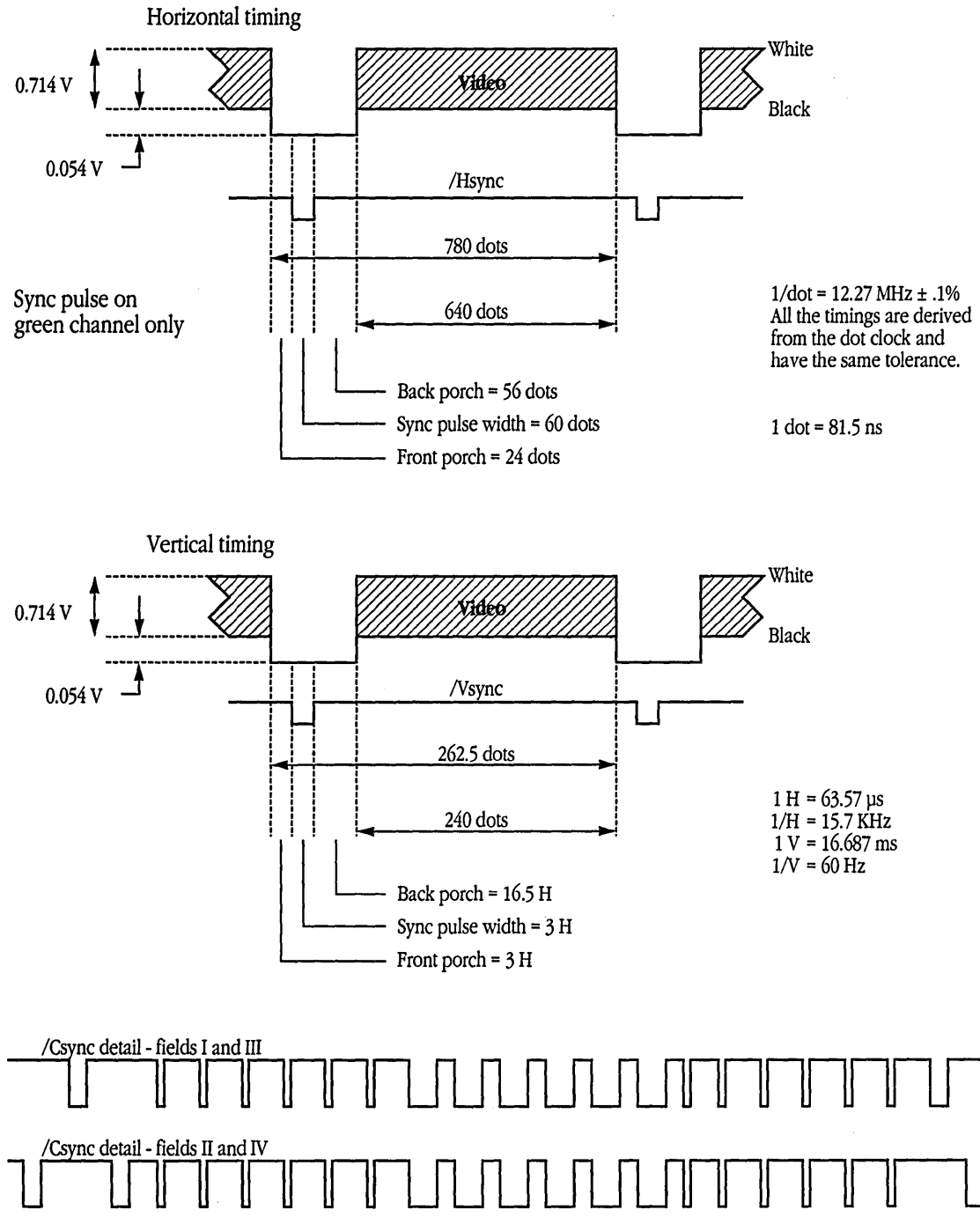
Horizontal and vertical scan timing

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



■ **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 Control Panel module called *Monitors*. (See the chapter on the Control Panel in *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 four 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.

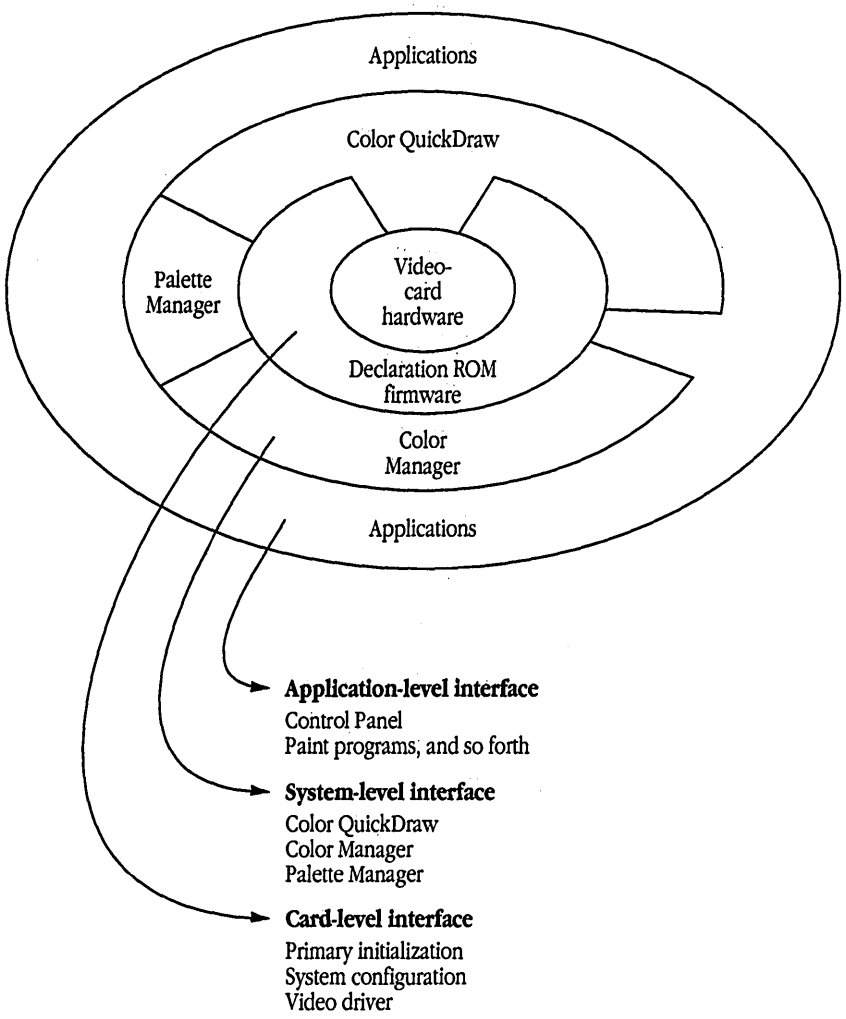
By making this determination at startup time, the primary initialization code permits the system code to be greatly simplified because only information pertinent to the connected monitor is reported by the Slot Manager, and no information about the other type of display is present in the Slot Device 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 chapter of *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 shown in Figure 6-1. The pinout of this connector is given in Table 5-3.

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 synch 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 five chapters give you the information you need to design expansion cards for Macintosh PDS computers.

Chapter 12 provides block diagrams of each of the Macintosh PDS computers, compares their major features, and gives a general overview of their operation. The chapter then introduces and describes the capabilities of the processor-direct slot expansion interface.

Chapter 13 contains the electrical information you need to design cards for the 68000 Direct Slot. The following topics are discussed in this chapter:

- electrical design guidelines for the Macintosh SE 68000 Direct Slot expansion interface including connector pinouts and signal descriptions, expansion card load limits and drive requirements, instructions for accessing the computer electronics, a map of available address space, and a power budget for each supply voltage.
- electrical design guidelines for the Macintosh Portable 68000 Direct Slot expansion interface including connector pinouts and signal descriptions, a functional description of the connector signals, and power allocation

Chapter 14 provides electrical guidelines for designing cards for the 68030 Direct Slot in the Macintosh SE/30 and the Macintosh IIfx. Topics include the 68030 Direct Slot expansion connector pinouts and signal descriptions; expansion connector load limits and drive requirements; information on accessing the computer main logic board, I/O, and memory devices from an expansion card; a discussion of pseudo-slot design; specific hints for developers; and power consumption guidelines.

Chapter 15 details the physical information you need to design Macintosh PDS expansion cards for each of the current Macintosh PDS computers.

Chapter 16 describes a proven design of a simple disk controller card that uses the Macintosh SE 68000 Direct Slot very successfully.

Appendix A, following Part II, provides information on electromagnetic interference (EMI), heat dissipation, and product safety standards.

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) for their expansion interface. Included in this category are the Macintosh SE, the Macintosh SE/30, and the Macintosh Portable. 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 and the Macintosh SE/30 with a 120-pin 68030 Direct Slot. Although the Macintosh IIfx also has a 120-pin 68030 Direct Slot, it is a member of the Macintosh II family of modular computers that use the NuBus as their primary expansion interface. The hardware overview of the Macintosh IIfx is provided in Chapter 1, "Hardware Overview of the Macintosh II Family."

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
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
Auxiliary processor	Not applicable	Power Manager IC Keyboard processor	Not applicable
Processor clock	7.8336 MHz	15.6672 MHz	15.6672 MHz
Coprocessor	Not applicable	Not applicable	MC68882 floating-point unit
Memory management	Not applicable	Not applicable	MC68030 includes a built-in PMMU that allows true 32-bit address translation with hardware page replacement
RAM	1 MB, expandable to 4 MB	1 MB, expandable to 5 MB using SRAM card	1 MB or 4 MB, expandable to 8 MB (expandable to 128 MB when higher density DRAM chips are available)
ROM	256 KB	256 KB	256 KB
Expansion slot	68000 Direct Slot, 96-pin	68000 Direct Slot, 96-pin	68030 Direct Slot, 120-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 Apple Desktop Bus (ADB) ports for keyboard, mouse, or optional input device

(Continued)

■ **Table 12-1** Major features of Macintosh computers with processor-direct slots (Continued)

Feature	Macintosh SE	Macintosh Portable	Macintosh SE/30
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
Floppy disk support	Integrated Woz Machine (IWM); controls two internal 3.5", 800 KB floppy disk drives (one standard, one optional)	Super Woz Integrated Machine (SWIM); controls two internal FDHD (1.4 MB, 3.5") drives (one standard, one optional); external FDHD drive port	Super Woz Integrated Machine (SWIM); controls internal FDHD (1.4 MB, 3.5") drive; external FDHD drive port
Hard disk	Optional internal 20 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
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
Sound	Standard Macintosh sound chip	Custom Apple Sound Chip (ASC)	Custom Apple Sound Chip (ASC)
Video display	Built-in 9" monochrome monitor, 512 x 342 pixels	Built-in LCD, 9.8" flat panel, 640 x 400 pixels	Built-in 9" monochrome monitor, 512 x 342 pixels
Battery	Long-life lithium battery backup	Rechargeable, 8-hour, lead-acid battery; retains RAM contents during sleep state	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.* Or if you are interested in a higher level overview, see the *Technical Introduction to the Macintosh Family.*

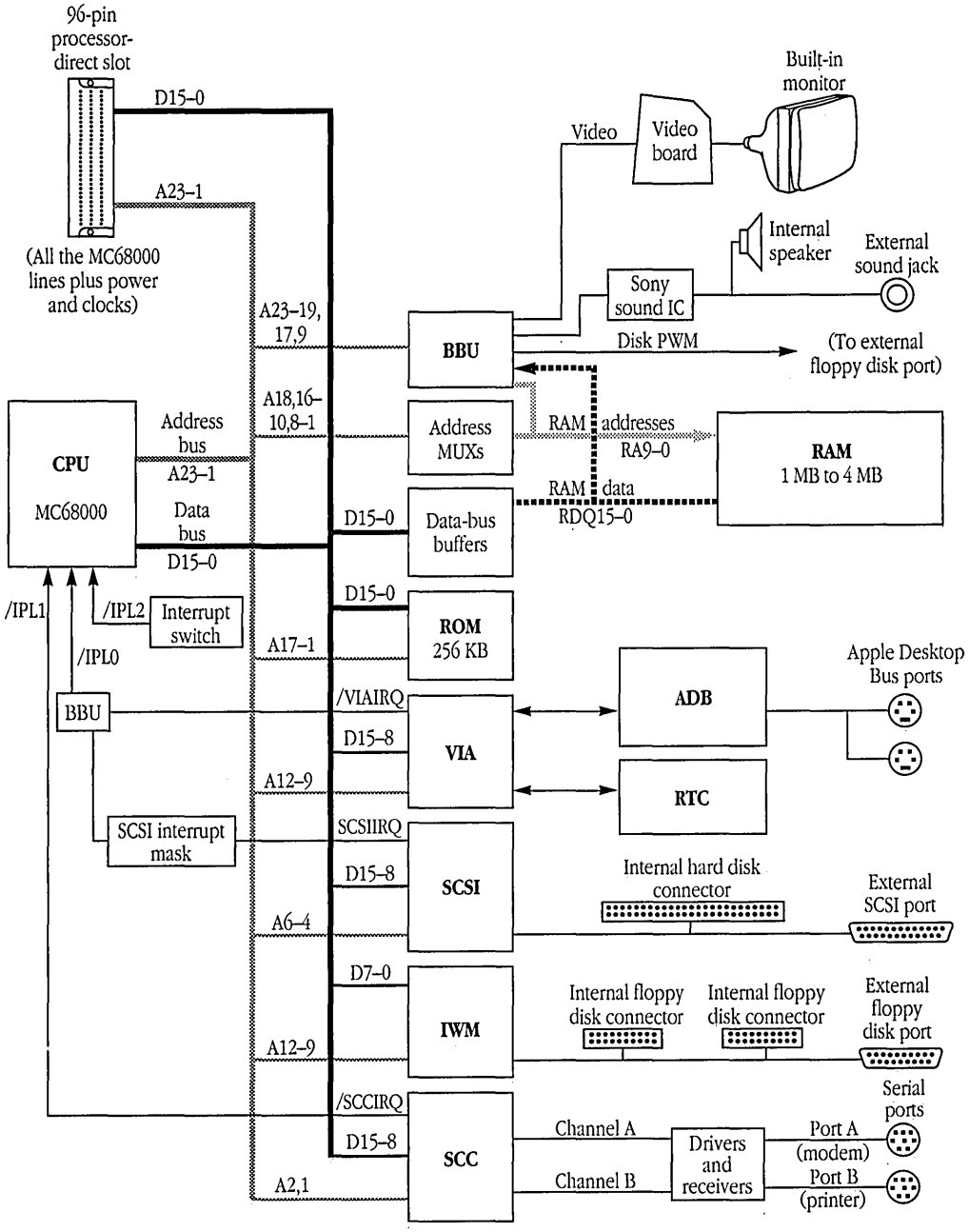
The Macintosh SE and the Macintosh SE/30 are similar in appearance to the original Macintosh computer. The Macintosh SE contains a Motorola MC68000 microprocessor operating at 7.8336 MHz and a Euro-DIN 96-pin connector for hardware expansion. The Macintosh Portable also has a Euro-DIN 96-pin expansion connector, but it is electrically different from the expansion connector used on the Macintosh SE. The Macintosh Portable differs from the Macintosh SE in that it has special low-power components throughout, including an MC68HC000 microprocessor operating at 15.6672 MHz, and incorporates a built-in liquid crystal display (LCD) flat panel for a display device.

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, and the Macintosh SE/30 are shown in Figures 12-1 through 12-3.

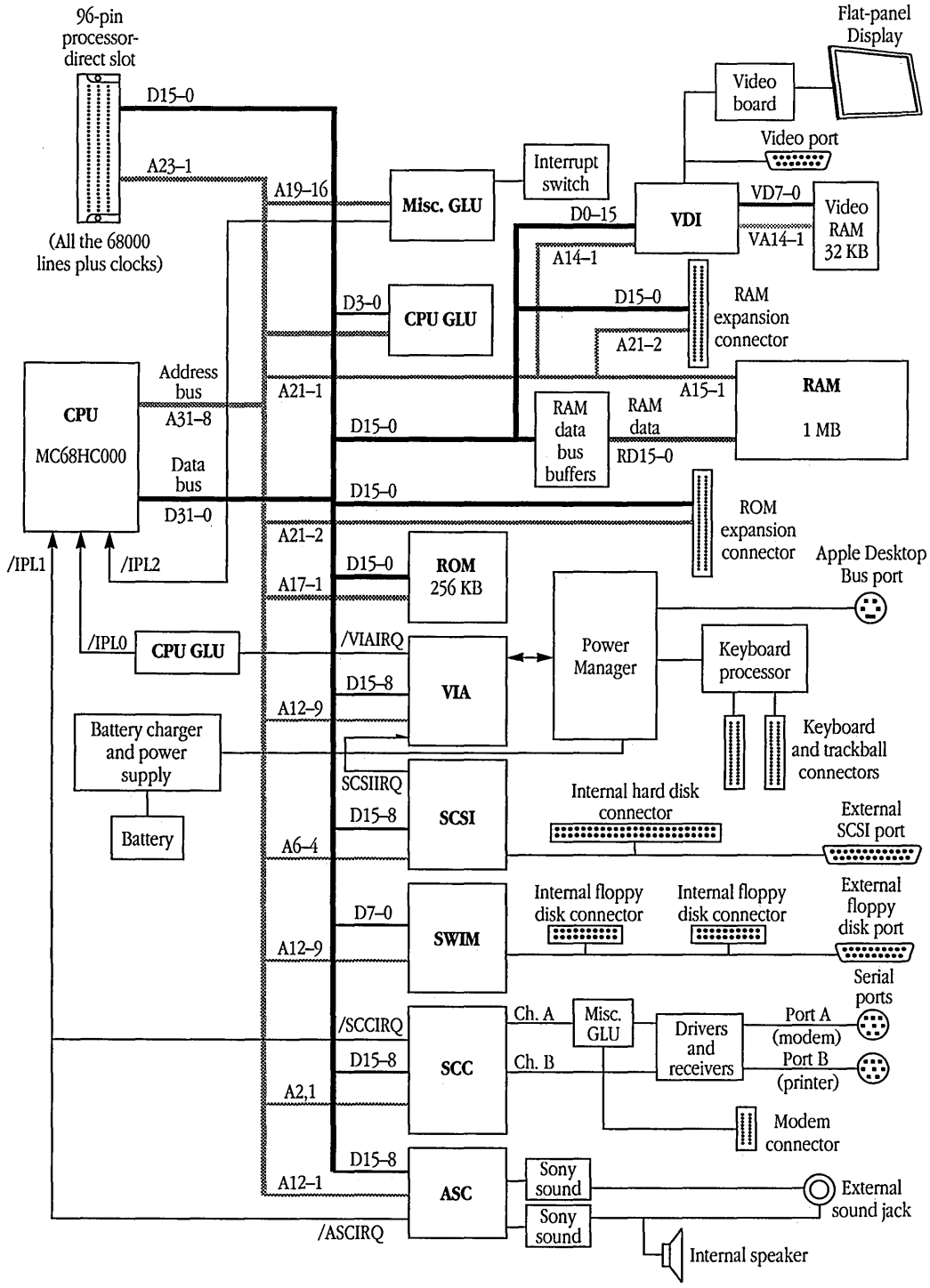
These processor-direct slot computers contain several common circuits including random access memory (RAM), read-only memory (ROM), and some I/O chips that enable the microprocessor to communicate with external devices. Following is a brief description of these I/O chips:

- Each Macintosh computer 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.
- An NCR 5380 SCSI (Small Computer System Interface) chip provides high-speed parallel communication with internal or external devices such as hard disks.
- A Zilog Z8530 Serial Communications Controller (SCC) provides for high-speed, asynchronous serial communication (also synchronous modem support).
- An Apple custom chip controls both internal and external floppy disk drives. The Macintosh SE uses an IWM (Integrated Woz Machine) chip to control 3.5-inch, 800 KB floppy disk drives; the Macintosh SE/30 and the Macintosh Portable use the SWIM (Super Woz Integrated Machine) chip to control 3.5-inch, 1.4 MB FDHD drives.

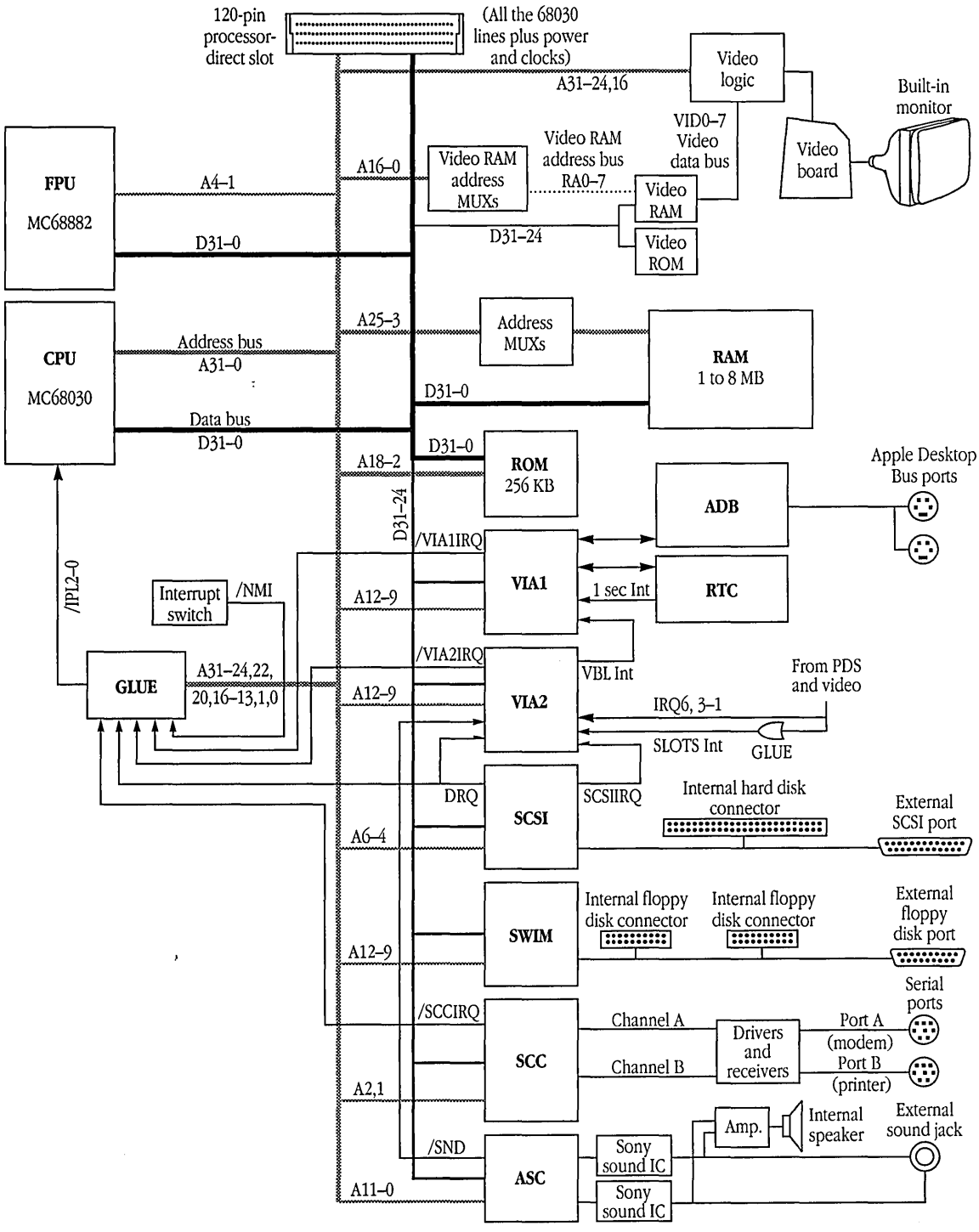
■ **Figure 12-1** Block diagram of the Macintosh SE



■ **Figure 12-2** Block diagram of the Macintosh Portable



■ **Figure 12-3** Block diagram of the Macintosh SE/30



- 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 assure that power will be applied during the access time.

All Macintosh computers use memory-mapped I/O, which means that you can gain access to each device in the system by reading 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 megabytes of address space. This address space is divided into several areas allocated to RAM, ROM, and various I/O devices. The MC68030 processor used in the Macintosh SE/30 can directly access 4 GB of address space, which is divided into several areas allocated to RAM, ROM, pseudo-slot expansion, and various I/O devices.

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. The actual amount of address space used depends upon the amount of RAM available in the system.

The MC68000 processor uses 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 first 256 bytes 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, and other information used by applications. For complete maps of the address ranges used in the Macintosh SE and the Macintosh Portable computers, see Figures 13-4 and 17-1, respectively. 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.

The MC68030 processor in the Macintosh SE/30 stores exception vectors in a 1024-byte-long table starting at the address of the vector base register. The first 256 bytes of the exception vector table are reserved for use by the operating system and the remainder are available for use by applications. The Macintosh SE/30 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.

In the Macintosh SE, the processor's accesses to RAM are interleaved with the video display's accesses. The average access rate to RAM of the Macintosh SE processor is about 3.22 MB per second. 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's used exclusively by the video display. The average access rate for the Macintosh SE/30 is about 15.6672 MB per second. The Macintosh Portable average access rate is about 6.45 MB per second.

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 as the constant romStart and is also 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

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.

Processor-direct slot interface

The processor-direct slot 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 16, 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 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.

◆ *Note:* Earlier editions of this book referred to the expansion interface in the Macintosh SE computer as the *SE Bus.* In this edition, the expansion interface is called the *68000 Direct Slot.*

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 of the Macintosh Portable case.

The PDS expansion interface on the Macintosh SE/30 consists of a Euro-DIN 120-pin connector that provides unbuffered access to the MC68030 processor bus. The larger pin count of the Macintosh SE/30 expansion connector allows you to take advantage of the more powerful features of the MC68030 processor used in that machine.

The 68000 Direct Slot

The 68000 Direct Slot expansion connector in the Macintosh SE provides unbuffered access to all of the signals from the microprocessor, including all address, data, and control signals. In addition, extensive power and grounding are provided, 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 four inches by eight 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 chassis have been designed to allow mounting and removal of the logic board while it is joined to an expansion card.

Chapter 15 describes the physical provisions for mounting an expansion card in a Macintosh SE 68000 Direct Slot. See Figures 15-2 and 15-5 for drawings of these mounting provisions.

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 "The Macintosh Portable 68000 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 Macintosh SE/30 is the first Macintosh computer with the 68030 Direct Slot expansion capability. Its microprocessor accesses both ROM and RAM at 15.6672 MB per second.

The design of the Macintosh SE/30 chassis simplifies card installation by allowing you to install the card vertically, rather than horizontally, as was the case in the Macintosh SE. This mounting configuration also provides more room, allowing you to design larger cards. You can install and remove your expansion card without removing the main logic board; you must, however, remove your expansion card before you can remove the main logic board. For more information on 68030 Direct Slot mounting provisions, see Chapter 15.

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 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 15 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 15 and Appendix A, use the Apple-supplied expansion features, and do not require physical alteration of the computer will not void the Apple Limited Warranty.

Motorola, Inc. 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, Inc. document AD1814R5, March 1985
- *MC68030 Enhanced 32-Bit Microprocessor User's Manual,* Motorola, Inc. document MC68030UM/AD

In summary, PDS expansion is supported by these features:

- Euro-DIN type expansion connector (96-pin on the Macintosh SE and Macintosh Portable, 120-pin on the Macintosh SE/30) that provides power, timing, and direct access to the computer's microprocessor bus
- stand-off mounting for card physical support
- high-capacity power supply and cooling fan (Macintosh SE and Macintosh SE/30)
- main logic board layout and installation features improved from earlier Macintosh models
- external device access opening (Macintosh SE and Macintosh SE/30) 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 budgets

The Macintosh SE 68000 Direct Slot

This section gives the pinouts and describes the signal characteristics of the 68000 Direct Slot expansion connector used on the Macintosh SE. This section also provides information and timing diagrams explaining how to access the computer's electronics from the expansion card, 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 Direct Slot expansion card are provided in Chapter 15.

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 pullup" in the table means that the /RESET line is normally driven open collector: it is *only* driven low, and a 1 kilohm pullup resistor on the main logic board returns the line to a logic 1.

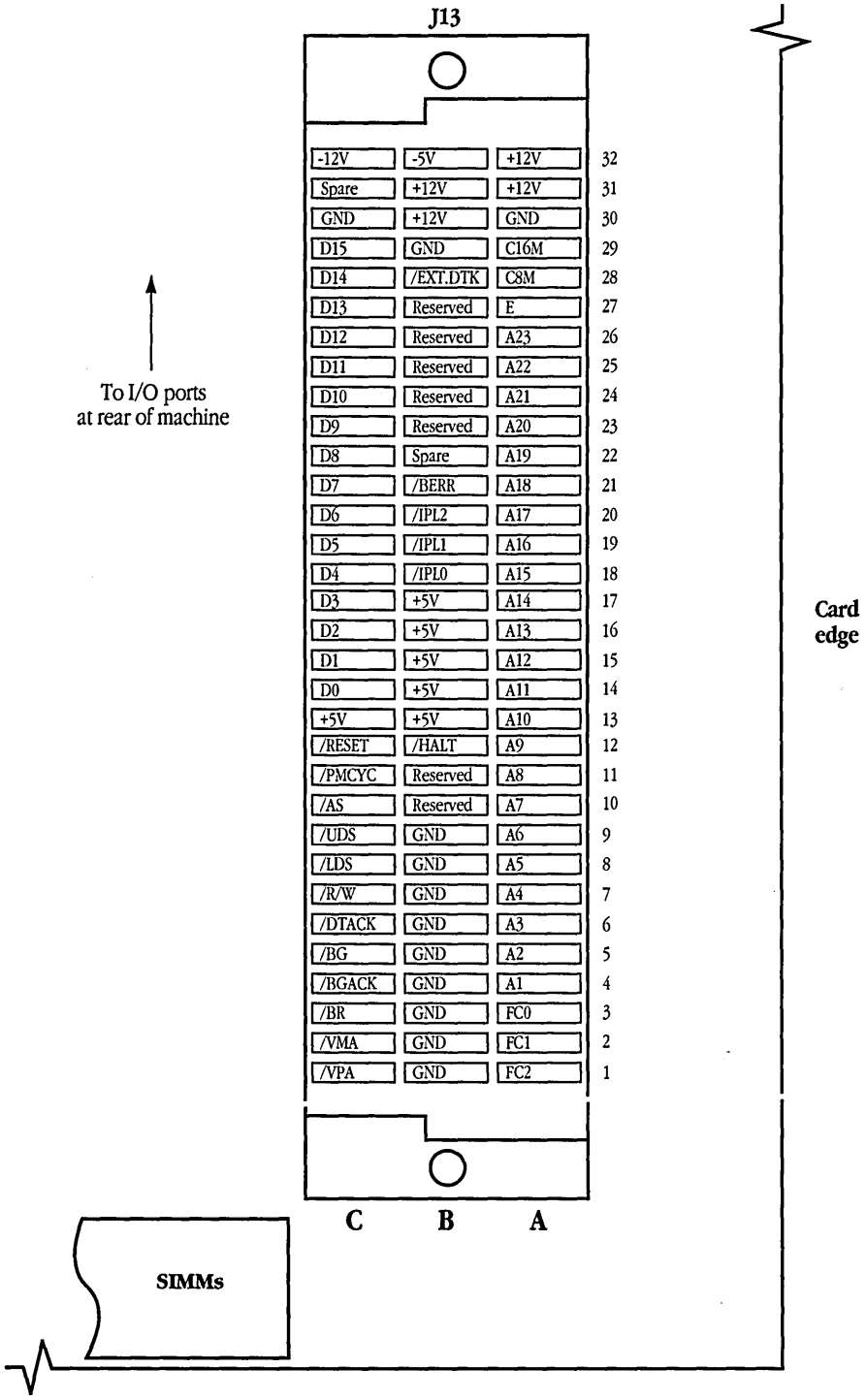
Correspondingly, /RESET presents a drive of 40 µA/.4 mA, 30 pF. This is the maximum amount of drive from the main logic board that is available to receiving integrated circuits on an expansion card. The /RESET line can drive an expansion card DC load of up to 40 µA in the high (logic 1) state, or up to .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, .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 two 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 tri-stated by the MC68000 after responding to a bus request. When tri-stated by the MC68000, this pin may be driven by an expansion card.

■ **Figure 13-1** Macintosh SE 68000 Direct Slot connector pinout



■ **Table 13-1** Macintosh SE 68000 Direct Slot signals, loading or driving limits

Connector Row	Pin	Signal name	Signal description	Input or output	Loading or driving limits (high/low)
A	1	FC2	Function code 2	Output (Input)	Drive: 40 µA/.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
A	2	FC1	Function code 1	Output (Input)	Drive: 40 µA/.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
A	3	FC0	Function code 0	Output (Input)	Drive: 40 µA/.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/.4 mA, 30 pF
A	27	E	E (enable) clock	Output	Drive: 40 µA/.4 mA, 30 pF
A	28	C8M	7.8336 MHz MC68000 clock	Output	Drive: 20 µA/.2 mA, 20 pF
A	29	C16M	15.6672 MHz gate array and IWM clock	Output	Drive: 20 µA/.2 mA, 20 pF
A	30	GND	Logic ground		
A	31	+12V	+12 volts	Output	Drive: 150 mA total, from all +12 V pins
A	32	+12V	+12 volts	Output	(see the section "Macintosh SE Power Budget")
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)
B	13–17	+5V	+5 volts	Output	Drive: 1.5 A total, from all +5 V pins (see the section "Macintosh SE Power Budget")

(Continued)

■ **Table 13-1** Macintosh SE 68000 Direct Slot signals, loading or driving limits (Continued)

Connector Row	Pin	Signal name	Signal description	Input or output	Loading or driving limits (high/low)
B	18	/IPL0	Interrupt level 0 (VIA, SCSI.IRQ)	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF (Open collector; 3.3 kΩ pullup)
B	19	/IPL1	Interrupt level 1 (SCC)	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF (Open collector; 3.3 kΩ pullup)
B	20	/IPL2	Interrupt level 2 (NMI switch)	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF (Open collector; 3.3 kΩ pullup)
B	21	/BERR	Bus error	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF (Open collector; 3.3 kΩ pullup)
B	22	Spare	Not connected		
B	23–27	Reserved	For future Apple use; do not connect		
B	28	/EXT.DTK	External /DTACK (tri-states main board's /DTACK)	Input	Load: 100 µA/2 mA, 50 pF (3.3 kΩ pullup)
B	29	GND	Logic ground		
B	30	+12V	+12 volts	Output	Drive: 150 mA total, from all +12 V pins
B	31	+12V	+12 volts	Output	(see the section "Macintosh SE Power Budget")
B	32	–5V	–5 volts	Output	Drive: 100 mA
C	1	/VPA	Valid peripheral address	Output	Drive: 40 µA/.4 mA, 30 pF
C	2	/VMA	Valid memory address	Output (Input)	Drive: 40 µA/.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 pullup)

(Continued)

■ **Table 13-1** Macintosh SE 68000 Direct Slot signals, loading or driving limits (Continued)

Connector Row	Pin	Signal name	Signal description	Input or output	Loading or driving limits (high/low)
C	4	/BGACK	Bus grant acknowledge	Input	Load: 100 µA/2 mA, 50 pF (3.3 kΩ pullup)
C	5	/BG	Bus grant	Output	Drive: 40 µA/.4 mA, 30 pF
C	6	/DTACK	Data transfer acknowledge	In/Out	Load: 100 µA/2 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF (3.3 kΩ pullup, /EXT.DTK low, tri-states main board's /DTACK)
C	7	R/W	Read/write	Output (Input)	Drive: 40 µA/.4 mA, 30 pF (Load: 200 µA/2 mA, 50 pF)
C	8	/LDS	Lower data strobe	Output (Input)	Drive: 40 µA/.4 mA, 30 pF (Load: 100 µA/1 mA, 50 pF)
C	9	/UDS	Upper data strobe	Output (Input)	Drive: 40 µA/.4 mA, 30 pF (Load: 100 µA/1 mA, 50 pF)
C	10	/AS	Address strobe	Output (Input)	Drive: 40 µA/.4 mA, 30 pF (Load: 200 µA/3.2 mA, 50 pF; 3.3 kΩ pullup)
C	11	/PMCYC	Processor memory cycle	Output	Drive: 40 µA/.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/.4 mA, 30 pF (Open collector; 1 kΩ pullup) (Connected to /HALT, pin B-12)
C	13	+5V	+5 volts	Output	Drive: 1.5 A total, from all +5 V pins (see the section "Macintosh SE Power Budget")
C	14–29	D0–15	Data bus, bits 0–15	In/Out	Load: 250 µA/1 mA, 100 pF Drive: 40 µA/.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
FC0–FC2	Function code lines.
A1–A23	Address lines.
E	E (enable) clock.
C8M	Microprocessor clock = 7.8336 MHz = C16M divided by 2.
C16M	Gate array clock = 15.6672 MHz.
/HALT	Halt. Wired directly to /RESET.
/IPL0–/IPL2	Interrupt priority level lines.
/BERR	Bus error. Generated by gate array due to SCSI access timeout. (Actually, /BERR is generated whenever /AS remains low for more than about 250 ms.)
/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).
/VPA	Valid peripheral address. Supplied by the gate array, coincident with /AS, for any access to VPA space ($E0 0000 to $FF FFFF).
/VMA	Valid memory address.
/BR	Bus request.
/BGACK	Bus grant acknowledge.
/BG	Bus grant.

(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 to $DF FFFF. If /AS falls after S3, /DTACK is supplied in S0 of the next access cycle (except for RAM accesses, which wait until S4 of the next cycle). /DTACK may be held off to wait for DRQ (DMA request from SCSI) in pseudo-DMA-mode SCSI accesses, to separate two successive accesses to the SCC, or to wait for a RAM access by video. /DTACK is not supplied for accesses to /VPA address space ($E0 0000 to $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 /DTACK to be externally generated by an expansion card.
R/W	Read/write.
/LDS	Lower data strobe.
/UDS	Upper data strobe.
/AS	Address 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.
D0–D15	Data bus.

Accessing the Macintosh SE electronics from an expansion card

An expansion card slave or 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 tri-states itself off the bus at the end of that bus cycle; 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 of the 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 (BBU, see Figure 12-1) 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 tri-state by pulling the /EXT.DTK line low.

When a device is accessed in the range $E0 0000 through $FF FFFF (the VIA, for example), 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 tri-state 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

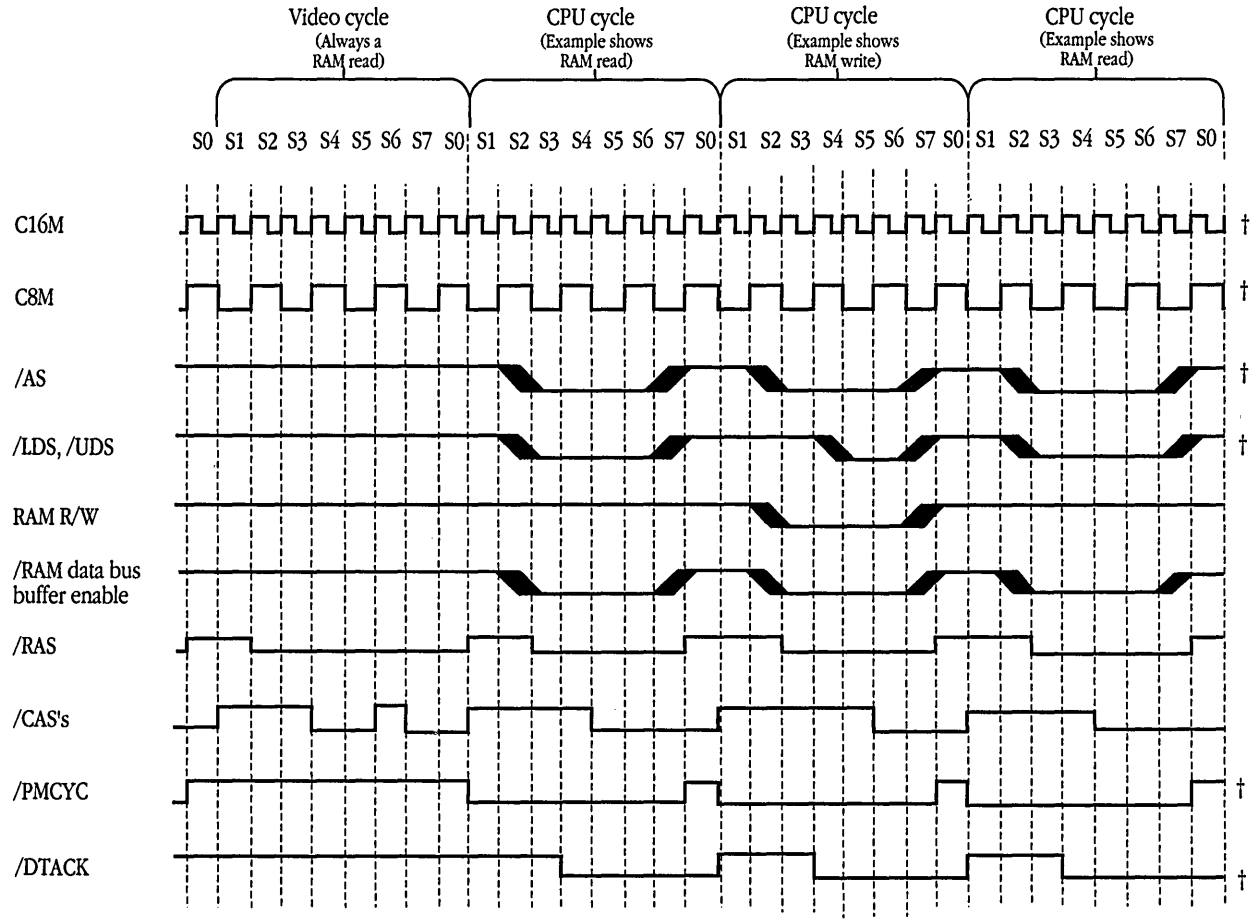
When an expansion card coprocessor accesses the RAM on the Macintosh SE logic board, the timing of these accesses is much more tightly constrained compared to accessing Macintosh I/O devices. Even if an expansion card coprocessor has its own on-card RAM, it will usually need to access the Macintosh SE 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.

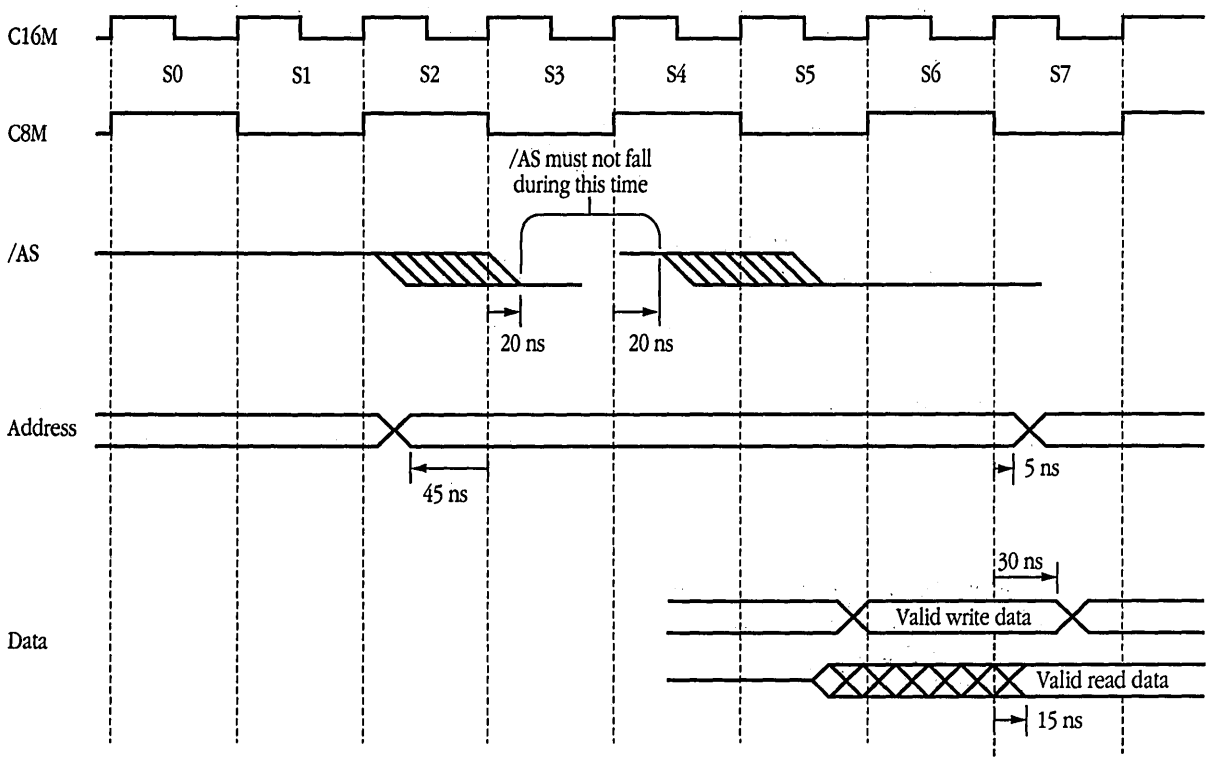
The BBU requires that a processor must not begin RAM accesses at random times. In normal operation, it expects any processor to behave more or less like an MC68000, which asserts /AS in S2 (see Figure 13-3 for details). The MC68000 in the Macintosh SE is automatically synchronized to the state machine 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 anytime 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.

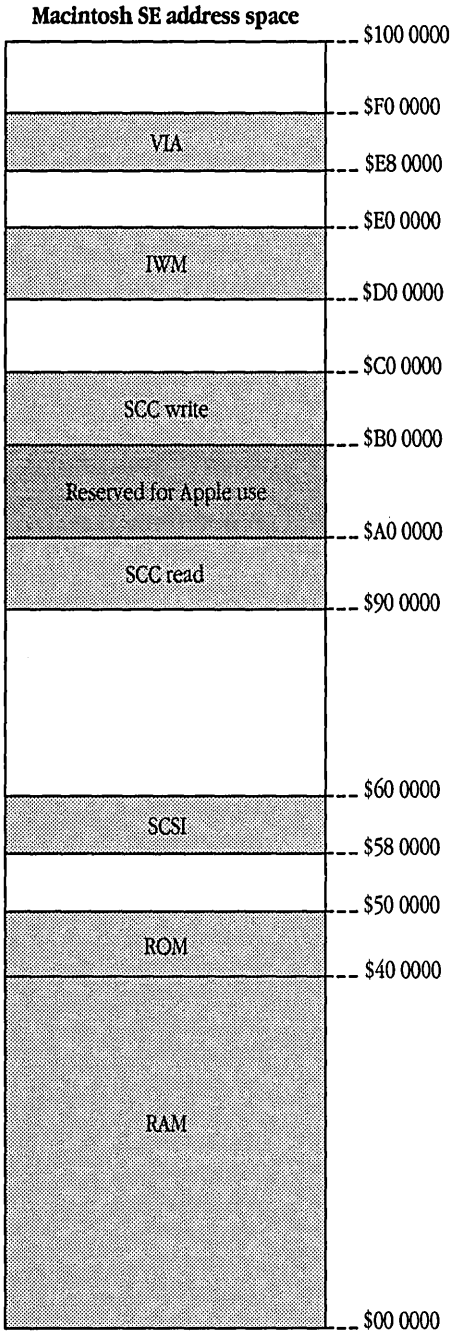
Available Macintosh SE address space

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 is 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 auto-vector location in low memory. The processor does an auto-vector 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 auto-vector reads.

■ **Figure 13-4** Macintosh SE address space



Macintosh SE power budget

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** 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	—	—	—
Internal SCSI hard disk	1.5	—	0.9	—
Expansion card†	1.5	0.1	0.15	0.1

† This is the allotted current for the expansion card, but, for thermal considerations, the total power of the expansion card should not exceed 7 watts.

The power budget specification for the 96-pin connector allows 1.5 A to be used from all +5 V 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 .15 A to be used from all +12 V pins combined. Peak surge current up to 1.5 A can be tolerated briefly (up to ten seconds)—when starting up a disk drive, for example.

The Macintosh Portable 68000 Direct Slot

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 15. In addition to its processor-direct slot, the Macintosh Portable contains connectors for ROM expansion cards and RAM expansion cards. The ROM and RAM expansion capability is described in Chapter 17.

△ **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 only a small amount of current for an 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. △

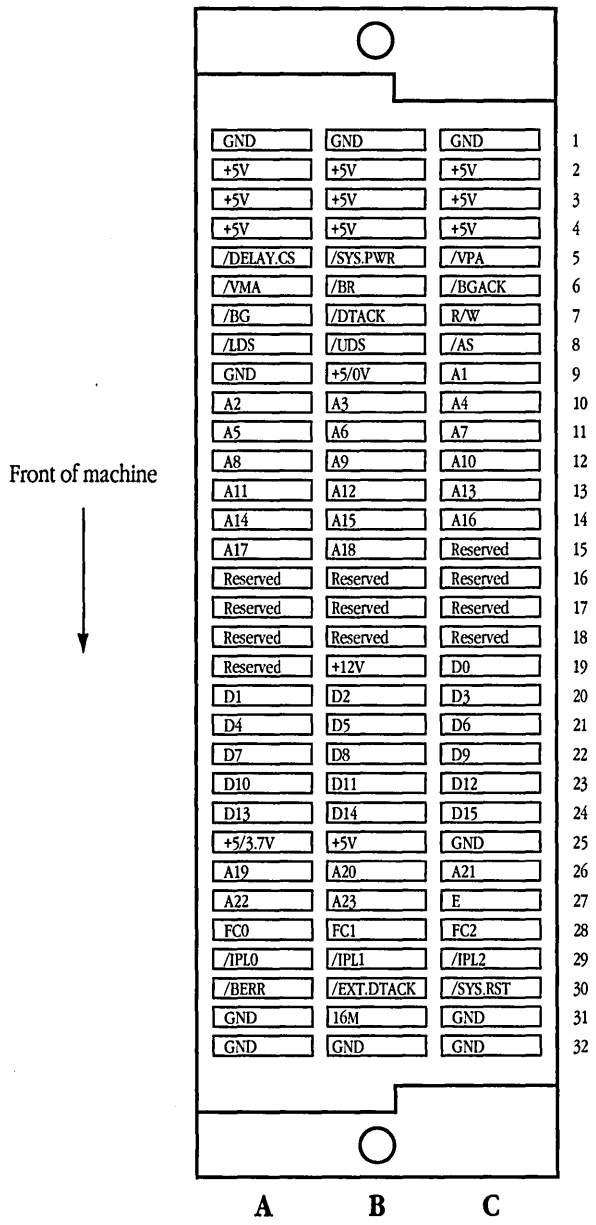
If you are determined to design an expansion card for the Macintosh Portable Direct Slot and can live within the aforementioned design limitations, you can contact Apple Macintosh Developer Technical Support (MacDTS) for additional guidance.

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



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
GND	Logic ground.
D0–D15	Unbuffered data bus, bits 0 through 15.
A1–A23	Unbuffered address bus, bits 1 through 23.
16M	16 MHz clock.
/EXT.DTACK	External data transfer acknowledge. This signal is an input to the processor logic glue that allows for external generation of the /DTACK signal.
E	E (enable) clock.
/BERR	Bus error signal generated whenever /AS remains low for more than about 250 ms.
/IPL0–/IPL2	Input priority level lines 0 through 2.
/SYS.RST	Initiates a system reset.
/SYS.PWR	A signal from the Power Manager IC that causes associated circuits to tri-state their outputs and go into the idle state; /SYS.PWR is pulled high (deasserted) during sleep state.
/AS	Address strobe.
/UDS	Upper data strobe.
/LDS	Lower data strobe.
R/W	Defines bus transfer as read or write signal.
/DTACK	Data transfer acknowledge.
/DELAY.CS	Input indicating that system is inserting wait states; can be used to gate chip selects.
/BG	Bus grant.
/BGACK	Bus grant acknowledge.
/BR	Bus request.
/VMA	Valid memory access.
/VPA	Valid peripheral address.

(Continued)

■ **Table 13-4** MC68HC000 signal descriptions (Continued)

Signal name	Description
FC0–FC2	Function code lines 0 through 2.
+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.
Reserved	For use by Apple.

Macintosh Portable power budget

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 fifty percent.

■ **Table 13-5** Macintosh Portable 68000 Direct Slot power budget

Power supply	Operating state	Sleep state
+5 V, always on	50 mA maximum	1 mA maximum
+5 V, switched	*	0 mA maximum
+12 V	25 mA maximum	0 mA maximum

* The 50 mA maximum applies to the loads of the switched and unswitched +5 V supplies.

Chapter 14 Electrical Design Guide for 68030 Direct Slot Expansion Cards

This chapter provides electrical guidelines for designing processor direct expansion cards for the Macintosh SE/30 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
- pseudo-slot expansion card design guidelines
- power consumption guidelines

About the 68030 Direct Slot

The 68030 Direct Slot expansion connector, first used on the Macintosh SE/30, takes advantage of the more powerful MC68030 microprocessor. In order to support the 32-bit address and data buses of the MC68030 microprocessor, the pin count of this connector was increased to 120 pins, as opposed to the 96-pin connector used on Macintosh MC68000-based machines to access the 16-bit address and data buses.

The 68030 Direct Slot is used primarily on compact, non-NuBus computers such as the Macintosh SE/30, but is also used on modular machines such as the Macintosh IIfx that have both NuBus and processor-direct slot interfaces.

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 following sections 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 and Macintosh IIfx computers. This information is followed by two more sections that give specific design guidelines for the Macintosh SE/30 expansion cards and the Macintosh IIfx expansion cards, respectively.

Electrical description of the Macintosh SE/30 68030 Direct Slot

Figure 14-1 gives the pinout for the 120-pin expansion connector on the Macintosh SE/30 main logic board, as viewed from above.

Table 14-1 lists the pin assignments, gives the signal names, and briefly describes each signal. Table 14-2 provides the load presented or drive available to each pin of an expansion card and indicates whether the signals are inputs or outputs.

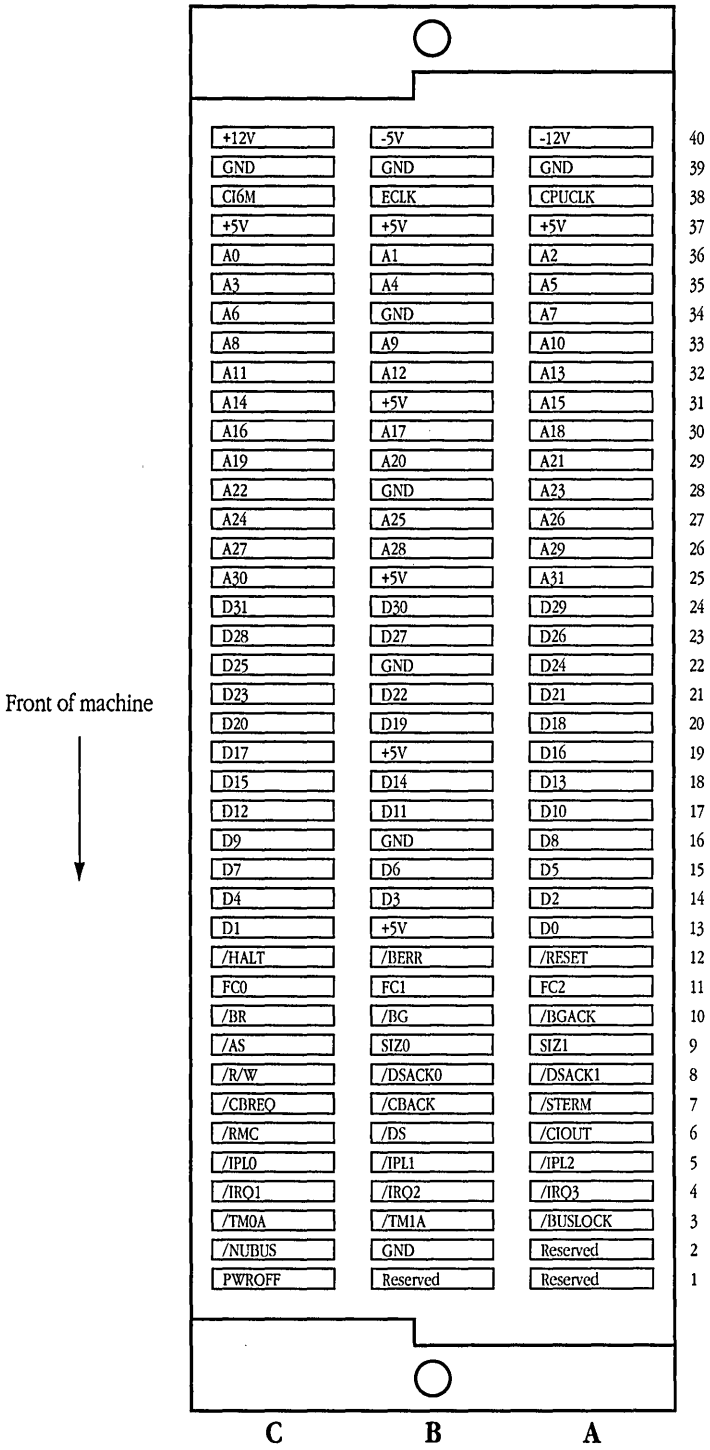
In the column labeled *Input/output* in Table 14-2, input refers to a signal from the expansion card to the processor and corresponds directly to the load shown in the column labeled *Load or drive limits*. Output refers to a signal from the processor to the expansion card and corresponds directly to 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 uA 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 pullup" in the table means that the /RESET line is normally in the open collector state; it is only driven low, and a 1 kilohm pullup resistor on the main logic board returns the line to a logic 1.

Additionally, /RESET presents a drive of 40 uA/.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 .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, .2 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.

Where "Load:" is in parentheses, the pin carries a signal that is usually an output driven by the MC68030 but that is tri-stated by the MC68030 after granting the bus to a DMA requester. When tri-stated by the MC68030, this signal may be driven by an expansion card.

■ **Figure 14-1** Macintosh SE/30 68030 Direct Slot connector pinout



■ **Table 14-1** Macintosh SE/30 68030 Direct Slot connector signals

Connector Row	Pin	Signal name	Signal description
A	1	Reserved	For use by Apple
A	2	Reserved	For use by Apple
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
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 14-1** Macintosh SE/30 68030 Direct Slot connector signals (Continued)

Connector Row	Pin	Signal name	Signal description
A	38	CPUCLK	15.6672 MHz CPU clock
A	39	GND	Ground
A	40	–12V	–12 volts
B	1	Reserved	For use by Apple
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
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

(Continued)

■ **Table 14-1** Macintosh SE/30 68030 Direct Slot connector signals (Continued)

Connector Row	Pin	Signal name	Signal description
B	36	A1	Address bit 1
B	37	+5V	5 volts
B	38	ECLK	E clock
B	39	GND	Ground
B	40	–5V	–5 volt
C	1	PWROFF	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
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

(Continued)

■ **Table 14-1** Macintosh SE/30 68030 Direct Slot connector signals (Continued)

Connector Row	Pin	Signal name	Signal description
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 gen clock
C	39	GND	Ground
C	40	+12V	+12 volts

■ **Table 14-2** 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/.4 mA, 30 pF
D0–D23	In/Out	Load: 150 µA/150 µA, 100 pF Drive: 40 µA/.4 mA, 30 pF
D24–D31	In/Out	Load: 300 µA/300 µA, 100 pF Drive: 20 µA/.2 mA, 30 pF
/RESET	In/Out	Load: 300 µA/8 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF Open collector, 1 kΩ pullup
/BERR	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF Open collector, 1 kΩ pullup
/HALT	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF Open collector, 1 kΩ pullup
FC0–FC2	Output (Input)	Drive: 20 µA/.2 mA, 30 pF (Load: 100 µA/8 mA, 50 pF) Open collector, 1 kΩ pullup
/BR	Input	Load: 100 µA/8 mA, 50 pF 1 kΩ pullup
/BG	Output	Drive: 40 µA/.4 mA, 30 pF

(Continued)

■ **Table 14-2** Macintosh SE/30 68030 Direct Slot signals, loading or driving limits (Continued)

Signal name	Input/output	Load or drive limits
/BGACK	Input	Load: 100 µA/8 mA, 50 pF 1 kΩ pullup
SIZ0–SIZ1	Output (Input)	Drive: 40 µA/.4 mA, 30 pF (Load: 100 µA/100 µA, 50 pF)
/AS	Output (Input)	Drive: 40 µA/.2 mA, 30 pF (Load: 100 µA/8 mA, 50 pF) Open collector, 1 kΩ pullup
/DSACK0–/DSACK1	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/.2 mA, 30 pF Open collector, 1 kΩ pullup
R/W	Output (Input)	Drive: 40 µA/.4 mA, 30 pF (Load: 100 µA/8 mA, 50 pF) Open collector, 1 kΩ pullup
/STERM	Input	Load: 100 µA/100 µA, 50 pF
/CBACK	Input	Load: 100 µA/100 µA, 50 pF
/CBREQ	Output	Drive: 40 µA/.4 mA, 30 pF
/CIOUT	Output	Drive: 40 µA/.4 mA, 30 pF
/DS	Output (Input)	Drive: 40 µA/.4 mA, 30 pF Load: 100 µA/8 mA, 50 pF Open collector, 1 kΩ pullup
/RMC	Output	Drive: 40 µA/.4 mA, 30 pF
/IPL0–/IPL2	In/Out	Load: 100 µA/100 µA, 50 pF Drive: 40 µA/.4 mA, 30 pF
/IRQ0–/IRQ3	Input	Load: 400 µA/4 mA, 50 pF
/TM0A	Input	Load: 400 µA/2 mA, 50 pF
/TM1A	Input	Load: 400 µA/2 mA, 50 pF
/BUSLOCK	Input	Load: 400 µA/2 mA, 50 pF
/NUBUS	Output	Drive: 40 µA/.4 mA, 30 pF
PWROFF	Output	Drive: 40 µA/.4 mA, 30 pF
CPUCLK	Output	Drive: 40 µA/.4 mA, 30 pF
C16M	Output	Drive: 40 µA/.4 mA, 30 pF
ECLK	Output	Drive: 40 µA/.4 mA, 30 pF

Electrical description of the Macintosh IIfx 68030 Direct Slot

Figure 14-2 gives the pinout for the 120-pin expansion connector on the Macintosh IIfx main logic board, as viewed from above.

Table 14-3 lists the pin assignments, gives the signal names, and briefly describes each signal. Table 14-4 provides the load presented or drive available to each pin of an expansion card and indicates whether the signals are inputs or outputs.

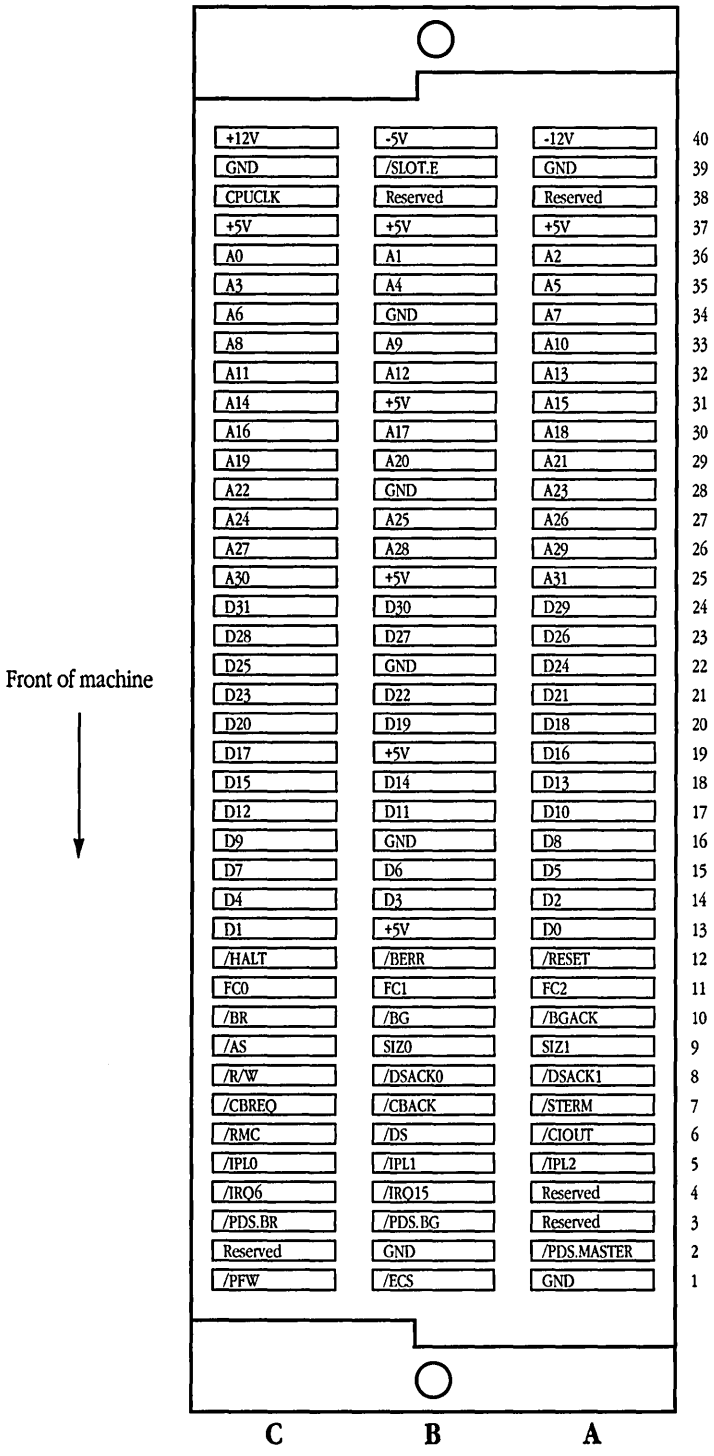
The last column in Table 14-4, labeled *Load or drive limits,* gives several specifications. An example may be helpful in interpreting this column. The /CBREQ line is shown as presenting a load of 100 µA/5 mA, 50 pF. This is the maximum expected load that an expansion card must drive when sending a /CBREQ 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 tri-states the signal and an expansion card may drive it. The notation "Tri-state, 1 kΩ pullup" gives the value of the required pullup resistor.

Additionally, /CBREQ presents a drive of 40 uA/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 pullup" in the table means that the /BERR line is normally in the open collector state; it is only driven low, and a 220 ohm pullup resistor on the main logic board returns the line to a logic 1. This is true for all open collector signals in Table 14-4.

Some of the expansion connector signals are specified to drive one 74LS input (a standard 74LS input load is 20 µA high, .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 14-2** Macintosh IIfx 68030 Direct Slot expansion connector pinout



■ **Table 14-3** Macintosh IIfx 68030 Direct Slot connector signals

Connector Row	Pin	Signal name	Signal 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 14-3** 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 pseudo-slot 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
B	35	A4	Address bit 4

(Continued)

■ **Table 14-3** Macintosh IIfx 68030 Direct Slot connector signals (Continued)

Connector Row	Pin	Signal name	Signal description
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 pseudo-slot $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
C	31	A14	Address bit 14
C	32	A11	Address bit 11

(Continued)

■ **Table 14-3** Macintosh IIfx 68030 Direct Slot connector signals (Continued)

Connector Row	Pin	Signal name	Signal description
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 14-4** 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
/RESET	In/out	Load: 18 µA, 260 pF Drive: 10 mA, 50 pF Open collector, 470 Ω pullup
/BERR	In/out	Load: 48 mA, 120 pF Drive: 6 mA, 15 pF (critical) Open collector, 220 Ω pullup
/HALT	In/out	Load: 48 mA, 100 pF Drive: 6 mA, 15 pF (critical) Open collector, 220 Ω pullup
FC0–FC2	(In)/out	(Load: 400 µA/4 mA, 80 pF) Drive: 80 µA/2.4 mA, 30 pF Tri-state, 3.3 kΩ pullup
/BR	In	Load: 100 µA/8 mA, 50 pF 3.3 kΩ pullup
/BG	Out	Drive: 40 µA/.6 mA, 30 pF

(Continued)

■ **Table 14-4** Macintosh IIfx 68030 Direct Slot signals, loading or driving limits (Continued)

Signal name	Input/output	Load or drive limits
/BGACK	In/out	Load: 10 mA, 100 pF Drive: 2 mA, 30 pF Open collector, 470 Ω pullup
SIZ0–SIZ1	(In)/out	(Load: 100 µA/100 µA, 100 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 Tri-state, 1 kΩ pullup
/DSACK0–/DSACK1	In/out	Load: 5 mA, 30 pF Drive: 10 mA, 50 pF Open collector, 1 kΩ pullup
R/W	(In)/out	(Load: 100 µA/5 mA, 150 pF) Drive: 80 µA/2.4 mA, 30 pF Tri-state, 1 kΩ pullup
/STERM	In/out	Load: 16 mA, 100 pF Drive: .6 mA, 15 pF (critical) Open collector, 330 Ω pullup
/CBACK	In/out	Load: 10 mA, 50 pF Drive: .6 mA, 30 pF Open collector, 470 Ω pullup
/CBREQ	(In)/out	(Load: 100 µA/5 mA, 50 pF) Drive: 40 µA/1.2 mA, 50 pF Tri-state, 1 kΩ pullup
/CIOUT	(In)/out	(Load: 40 µA/1.6 mA, 50 pF) Drive: 20 µA/.6 mA, 50 pF Tri-state, 3.3 kΩ pullup
/DS	(In)/out	(Load: 100 µA/5 mA, 100 pF) Drive: 80 µA/2.4 mA, 50 pF Tri-state, 1 kΩ pullup
/RMC	(In)/out	(Load: 40 µA/1.6 mA, 50 pF) Drive: 20 µA/.6 mA, 50 pF Tri-state, 3.3 kΩ pullup
/IPL0–/IPL2	Out	Drive: 40 µA/.4 mA, 30 pF
/PFW	Out	Refer to the section "/PFW Interaction With the Power Supply" in Chapter 5 for details.

(Continued)

■ **Table 14-4** Macintosh IIfx 68030 Direct Slot signals, loading or driving limits (Continued)

Signal name	Input/output	Load or drive limits
CPUCLK	Out	Drive: 80mA; this signal is driven by an emitter follower.
/PDS.MASTER	In	Load: 25 µA/250 µA, 50 pF
/PDS.BG	In	Load: 25 µA/250 µA, 50 pF 3.3 kΩ pullup
/PDS.BR	Out	Drive: 100 µA/8 mA, 50 pF
/ECS	(In)/out	(Load: 100 µA/5 mA, 50 pF Drive: 20 µA/.6 mA, 15 pf (critical) Tri-state, 1 kΩ pullup
/SLOT.E	In	Load: 25 µA/250 µA, 50 pF 3.3 kΩ pullup

Functional description of the MC68030 signals

The Macintosh SE/30 and future MC68030-based PDS machines without NuBus will share a common set of address, data, control, power, and ground signals. This means that the Direct Slot pin assignments for these machines will be identical to the Macintosh SE/30.

The Macintosh SE/30 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 14-5 lists those signals that are common to the two machines as well as future MC68030-based PDS machines without NuBus. Two of the clock signals (ECLK and C16M) shown in Table 4-5 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 future Macintosh PDS computers without NuBus will share a common set of these machine-specific signals. Macintosh computers such as the Macintosh IIfx that have both NuBus and the 68030 Direct Slot will probably each have a unique set of machine-specific signals. Pins that are currently defined as reserved (see Figures 14-1 and 14-2) may be added to the lists of machine-specific signals on future machines.

■ **Table 14-5** MC68030 common signals on the 68030 Direct Slot

Signal name	Description
A0–A31	Address lines.
D0–D31	Data lines.
/IPL0–/IPL2	Interrupt priority level lines. *
/CIOUT	A tri-state output signal that inhibits the operation of an external cache. †
/CBACK	An input signal indicating that the accessed device can operate in burst mode.
/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. ‡
/DSACK0–/DSACK1	Data transfer acknowledge signals that indicate the completion of a data transfer operation.
SIZ0–SIZ1	Tri-state output signals indicating the number of bytes remaining to be transferred during the current bus cycle.
/BGACK	An input signal indicating that an external device has become bus master.
FC0–FC2	Three-bit function code used to identify the address space of current bus cycle.
/RESET	A bi-directional signal that initiates a system reset.
/BG	An output signal indicating that an external device may become bus master following completion of the current processor bus cycle.
/BERR	A bus error signal indicating that an invalid bus operation is being attempted.‡
ECLK	Main logic board VIA chip clock signal (not used on the Macintosh IIfx).

(Continued)

■ **Table 14-5** MC68030 common signals on the 68030 Direct Slot (Continued)

Signal name	Description
/RMC	A tri-state output signal that identifies the current bus cycle as part of an indivisible read-modify-write operation.
/CBREQ	A tri-state output signal indicating a burst request for the instruction or data cache.
R/W	A tri-state output signal that defines the bus transfer as a read or write cycle.
/AS	A tri-state output signal indicating that a valid address is on the bus.
/BR	An input signal indicating that an external device wishes to become bus master.
/HALT	A signal indicating that the processor should suspend bus activity.‡
/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.
C16M	A general purpose 15.6672 MHz clock (not used on the Macintosh IIfx).

* 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.

† On the Macintosh IIfx, /CIOUT must not be used in conjunction with /CBREQ because the cache should not be inhibited during burst mode cycles.

‡ The maximum capacitive load allowed on these signal lines is 15 pF due to the high-speed nature of the Macintosh.

Macintosh SE/30 68030 Direct Slot machine-specific signals

Table 14-6 lists the 68030 Direct Slot signals that are specific to the Macintosh SE/30 computer. All of the machine-specific signals listed in Table 14-6 (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 computer, you should design your 68030 Direct Slot expansion card to occupy the same 32-bit physical address ranges occupied by NuBus cards in Macintosh II–family computers. This method of emulating NuBus expansion slot address space is called **pseudo-slot design.** The pseudo-slot interrupt support lines allow the use of the Macintosh Slot Manager driver routines and thus provide an easy software port for NuBus designs. Pseudo-slot design is the preferred expansion design strategy for Macintosh computers with processor-direct slots but without NuBus. See the section "Pseudo-Slot Design Guidelines for Macintosh SE/30 PDS Expansion Cards," later in this chapter, for more information on pseudo-slot design.

Cards that take advantage of these pseudo-slot features won't work in a Macintosh NuBus slot because of bus conflicts with physical NuBus. These pseudo-slot signal lines will be available on future 68030-based Macintosh PDS computers without physical NuBus. A slightly different pseudo-slot signal configuration is used on machines that include both the NuBus expansion interface and a processor-direct slot.

By porting the NuBus design to the 68030 Direct Slot via pseudo-slot, 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 14-6** Macintosh SE/30 machine-specific signals on the 68030 Direct Slot

Signal name	Description
PWROFF	Status signal to inform an expansion card that power will soon be removed. (Shutdown has been selected from the Special menu.) This signal is common across Macintosh machines without physical NuBus.
/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.

(Continued)

■ **Table 14-6** Macintosh SE/30 machine-specific signals on the 68030 Direct Slot (Continued)

Signal name	Description
/IRQ1–/IRQ3	General purpose interrupts that correspond to the three pseudo-slot addresses. These signals are common across Macintosh machines without physical NuBus.
/TM0A–/TM1A	Status input signals to VIA2 that are currently not used.
/NUBUS	Address decode of the memory range $6000 0000 to $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.
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 may change with each new Macintosh.

Macintosh IIfx 68030 Direct Slot machine-specific signals

Table 14-7 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 pseudo-slot 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 pseudo-slot $E, while the Macintosh SE/30 uses three interrupt lines to support its NuBus pseudo-slot addresses. The Macintosh IIfx also includes another interrupt line, /IRQ15, that you should use if your design does not support the NuBus pseudo-slot and you are providing a stand-alone, card-specific software driver.

■ **Table 14-7** Macintosh IIfx machine-specific signals on the 68030 Direct Slot

Signal name	Description
/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.
/SLOT.E	When active, this signal indicates that the PDS expansion card is replacing NuBus slot $E in the Macintosh IIfx address map.
/PDS.MASTER	When this signal is active, the PDS expansion card replaces the MC68030 in the bus arbitration scheme.
/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.
/ECS	A signal from the MC68030 indicating early cycle start.
/IRQ6	A dedicated interrupt line, from the processor to the 68030 Direct Slot, that supports NuBus pseudo-slot $E. To prevent incompatibility on the Macintosh IIfx, use /IRQ6 instead of input priority level lines /IPL0 through /IPL2.
/IRQ15	An interrupt line that is used if the expansion card does not support the NuBus pseudo-slot.

Design considerations for Macintosh SE/30 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 pseudo-slot design, a description of how the interrupt handling mechanism works, a summary of design hints, and a discussion of a Macintosh SE/30 expansion card's 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 14-8 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.

In comparison to 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 tri-states 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 GLUE 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 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 14-8** 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 pseudo-slot is not used)
$6000 0000–$F8FF FFFF	Expansion (undefined)
$F900 0000–$FBFF FFFF	Expansion pseudo-slot 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 14-8 is valid.

Pseudo-slot design guidelines for Macintosh SE/30 expansion cards

If you are familiar with designing devices for the MC68000 family of microprocessors, you should find it relatively easy to use the pseudo-slot method to design an expansion card for the Macintosh SE/30. 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 II-family products. 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.

If you do not use pseudo-slot design, the area from $5800 0000 to $5FFF FFFF in the Macintosh SE/30 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 chapter of *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 14-9.

■ **Table 14-9** 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 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 expansion slot provides three general-purpose interrupt inputs to the main logic. These interrupts correspond to the first three NuBus slots of a Macintosh II. In order 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 14-10 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 QuickDraw with your card. △

In order 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 pseudo-slot 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 14-10** Pseudo-slot address ranges for Macintosh SE/30 expansion cards

Interrupt	32-bit space address
1	$F900 0000—$F9FF FFFF
2	$FA00 0000—$FAFF FFFF
3	$FB00 0000—$FBFF FFFF

Interrupt handling for the Macintosh SE/30 68030 Direct Slot

The interrupt handling mechanism for the 68030 Direct Slot on the Macintosh SE/30 is similar to the mechanism used in the Macintosh II family. 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 auto-vectored 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 pseudo-slot 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 auto-vectored, 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 (auto-vector) pin. Once the processor determines the AVEC pin is signaling an auto-vector, 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 expansion cards

When designing a card for the Macintosh SE/30, 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 main logic board that generates a /BERR signal anytime the address strobe is asserted for longer than 44 microseconds. 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, but this may not be true for future 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 main logic board. The timing of the data strobe should match the MC68030 data strobe signal.

Notice that the /NUBUS signal (Table 14-6) is an address decode of the memory range $6000 0000 to $FFFF FFFF. The /AS (address strobe) signal qualifies the assertion of the /NUBUS signal. The /NUBUS signal is asserted a maximum of 26 nanoseconds after the /AS signal is asserted, and is removed a maximum of 22 nanoseconds 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 pseudo-slot address ranges when using the /NUBUS signal as a qualifier.

The pseudo-slot 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 Macintosh SE/30.

Power consumption guidelines for Macintosh SE/30 expansion cards

The Macintosh SE/30 uses 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 "Macintosh SE Power Budget." The Macintosh SE/30 main logic board consumes 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 14-11 shows the allotted current for an expansion card.

◆ *Note:* For thermal considerations, the total power of the expansion card should not exceed 7 watts.

■ **Table 14-11** Power budget for a Macintosh SE/30 expansion card

Device	+5 V	–5 V	+12 V	–12 V
Expansion card	1.5 amps	0.1 amps	0.15 amps	0.1 amps

△ **Important** You should seriously consider whether routing power outside the case is necessary. If power is required outside the case, all pins carrying power must be protected by a fuse against an overcurrent load. You should use 1-ampere fast-acting fuses to retain the product safety compliance designed into the Macintosh SE/30. △

Design considerations for Macintosh IIfx PDS expansion cards

This section provides technical information that you need to design a PDS expansion card for the Macintosh IIfx computer. Topics covered include pseudo-slot design, termination of memory cycles, interrupt handling mechanism, bus priority scheme, the effect of clock speeds on expansion card design, the use of the cache memory, and power consumption guidelines.

Pseudo-slot design guidelines for Macintosh IIfx PDS expansion cards

It is relatively easy to use the pseudo-slot 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 processor-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 pseudo-slot design, your expansion card can occupy either the slow slot space area ($6000 0000 to $6FFF FFFF) or the fast slot space area ($7000 0000 to $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 microseconds. 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 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 a 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 used the pseudo-slot 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 pseudo-slot design method, all interrupts on the /IRQ15 line are fielded as nonslot $E interrupts.

Bus master priority scheme for Macintosh IIfx

You should be aware of the fact that 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 14-12. Note that the NuBus and SCSI interfaces allow DMA access to the 68030 Direct Slot.

■ **Table 14-12** 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 is 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 pseudo-slot address space $Exxx xxxx or $FExx xxxx. It also shifts speed if you do not use pseudo-slot 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.

An area of possible concern in some systems is the possibility of thrashing that could 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 byte, word, misaligned words, and longwords, but the processor must execute multiple cycles to gain access to code written in this manner. Also, it is important that, if possible, you 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. /DS 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.

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.

Chapter 15 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 SE/30 (68030 Direct Slot), and the Macintosh IIfx (68030 Direct Slot) computers.

The information includes

- mechanical drawings showing expansion card dimensions and mounting provisions
- descriptions of the mating 96-pin and 120-pin connectors on the expansion cards and logic boards
- mechanical drawings detailing electrical and physical requirements for connecting expansion cards to external equipment

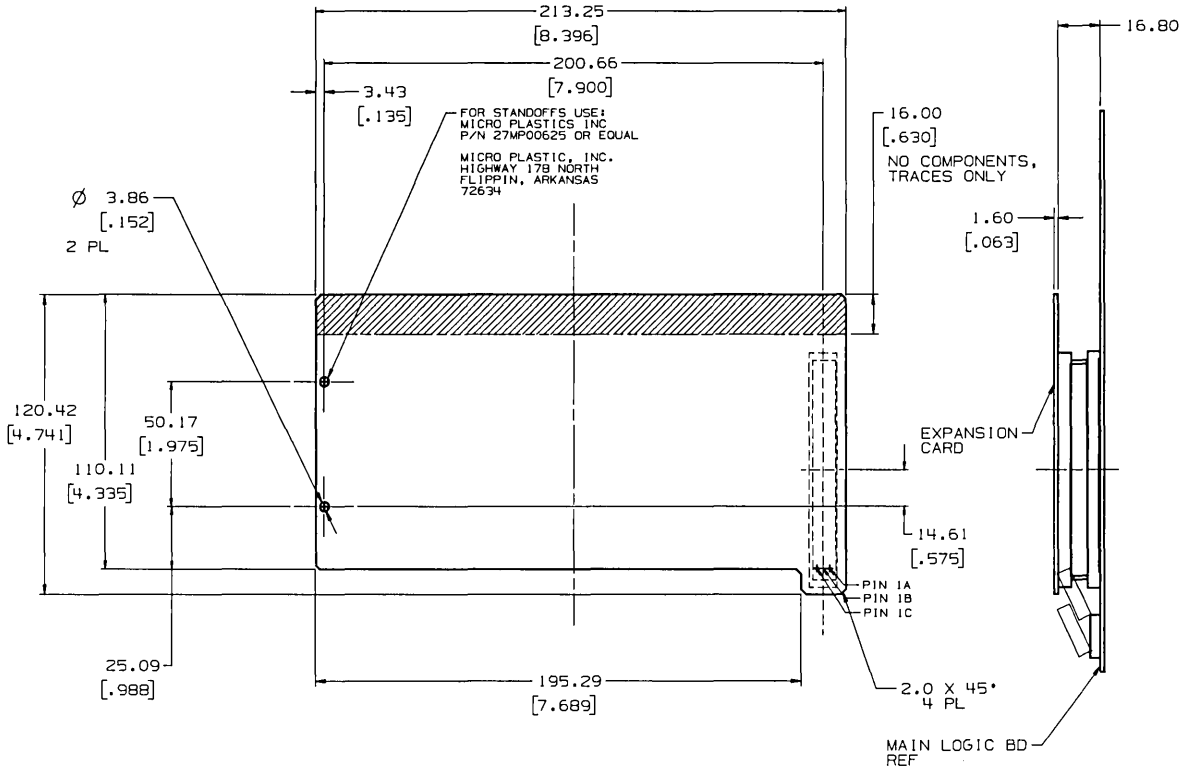
▲ **Warning** The drawings in this chapter are from mechanical design guides used within Apple Computer. They were correct at the time of publication but are subject to change in the future. ▲

Physical guidelines for Macintosh SE expansion cards

Figures 15-1 through 15-3 show the spatial relationship between an expansion card and the Macintosh SE main logic board.

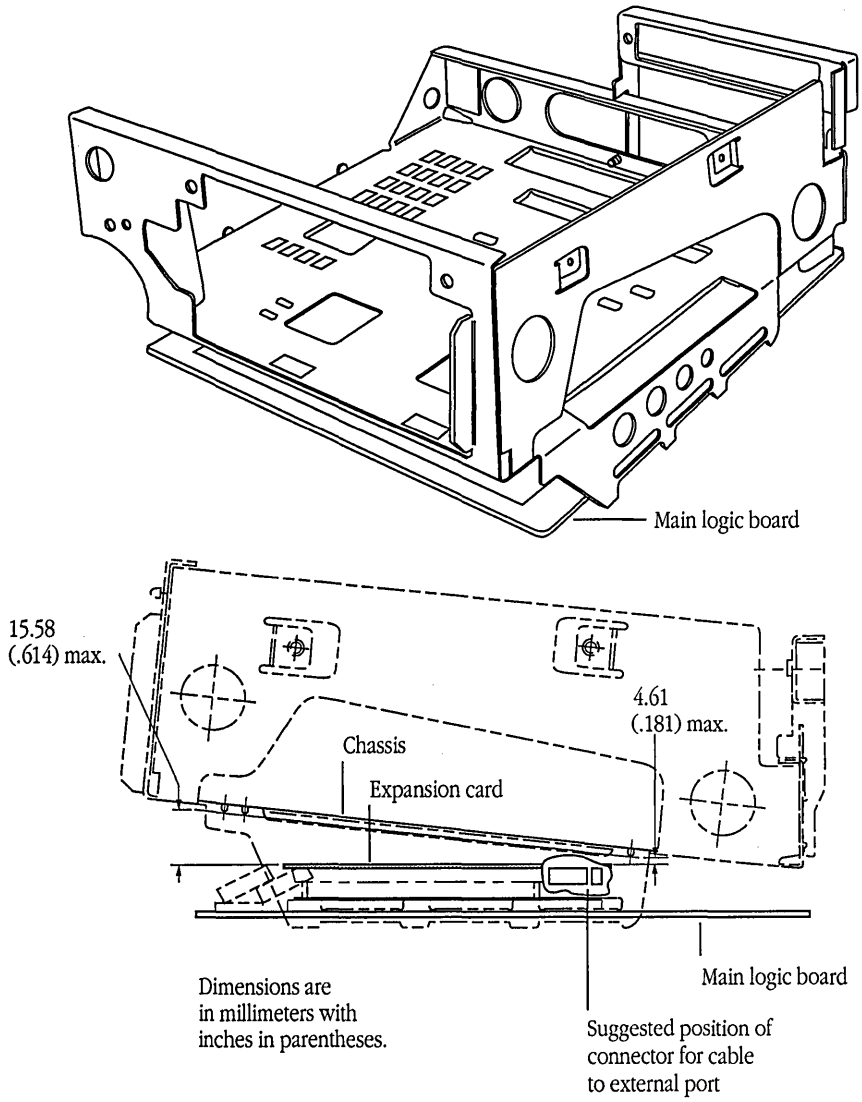
▲ **Warning** Figure 15-1 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change in the future. ▲

■ **Figure 15-1** Macintosh SE expansion card design guide



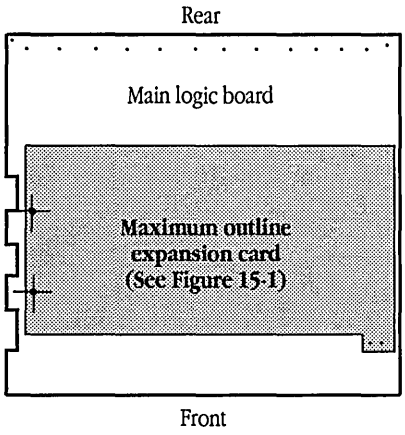
COMPONENT (TOP) VIEW
COMPLIANT PINS ON DIN 96

■ **Figure 15-2** An expansion card in the Macintosh SE assembly



Side view height restrictions

■ **Figure 15-3** An expansion card and the Macintosh SE main logic board



The 68000 Direct Slot 96-pin connector for the Macintosh SE

Figure 15-4 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.

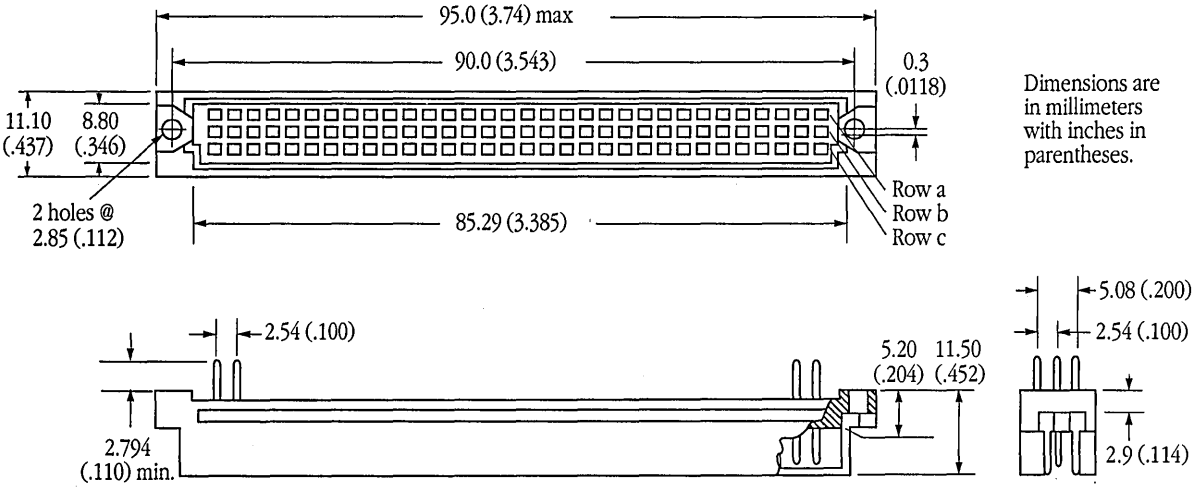
Figure 15-5 shows the 96-pin socket connector and mounting supports on the Macintosh SE main logic board assembly. Figure 15-6 is a detail of the socket connector used on the main logic board.

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.

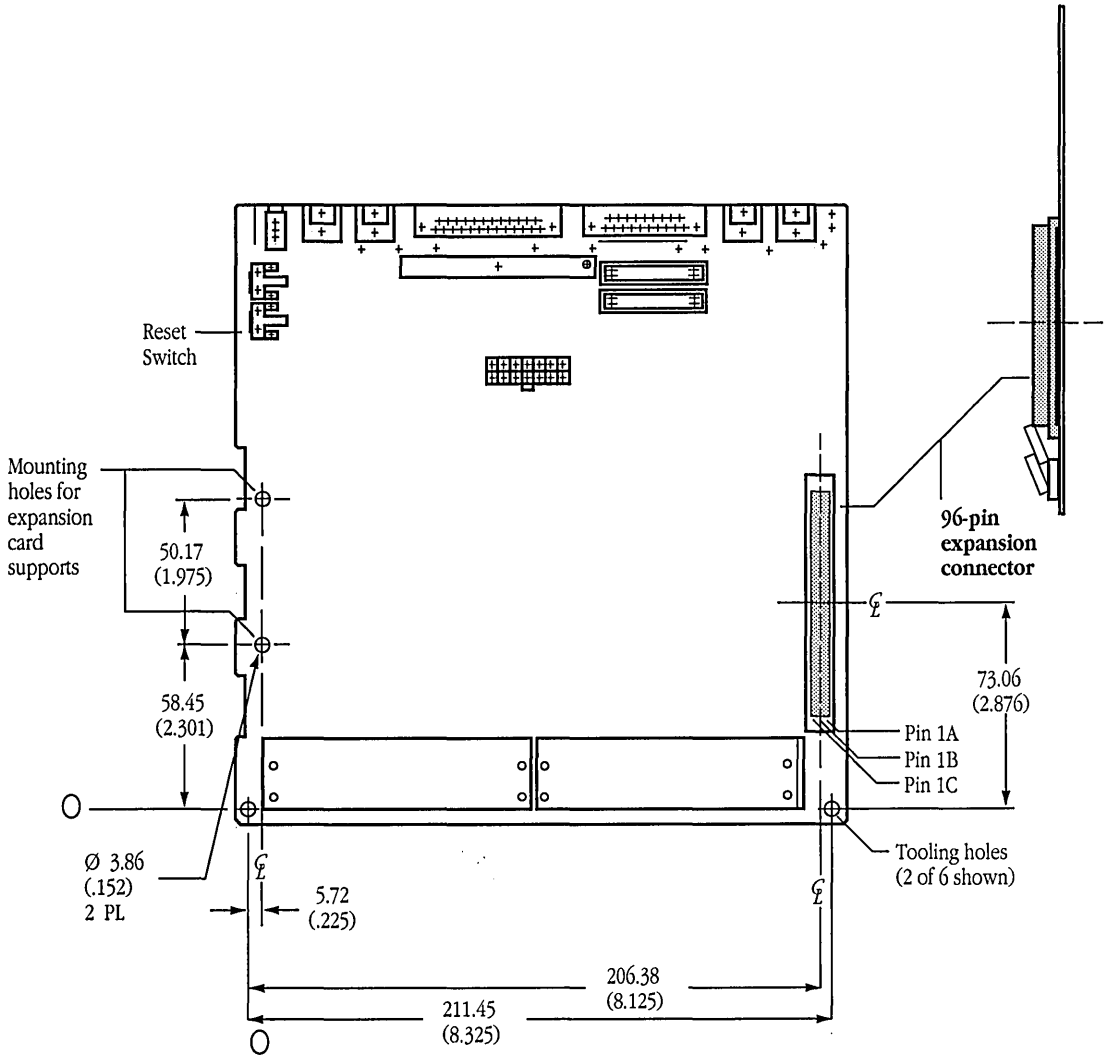
■ **Figure 15-4** 96-pin plug connector for a Macintosh SE expansion card



Three-row pin connector

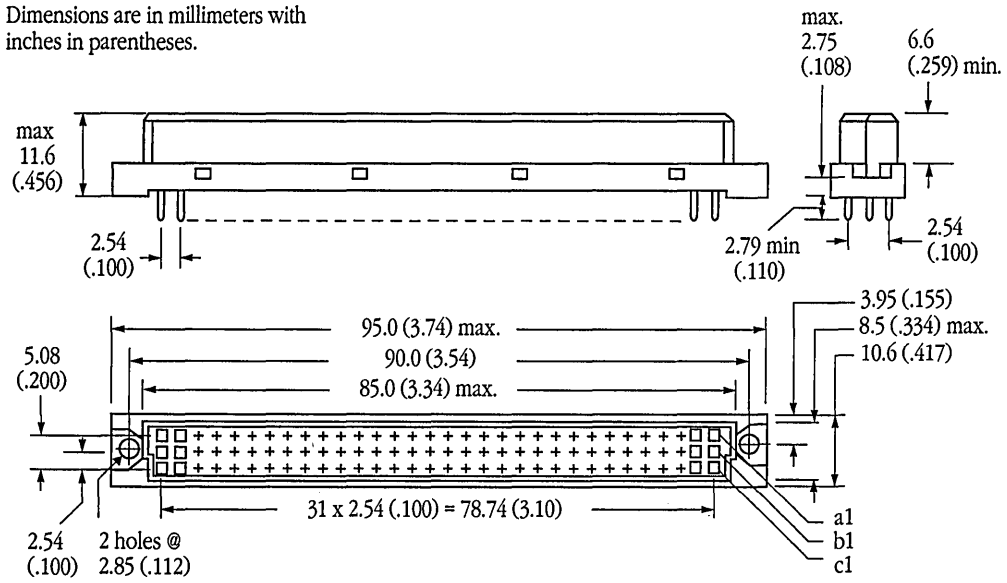
96 contact positions
2.54 mm (.100 inch) spacing pins
Gold plated, 20 microinches, over nickel plate

■ **Figure 15-5** Macintosh SE connector and mounting supports for an expansion card



Dimensions are in millimeters
with inches in parentheses.

■ **Figure 15-6** Detail of 96-pin socket connector used on Macintosh SE main logic board



Three-row socket connector

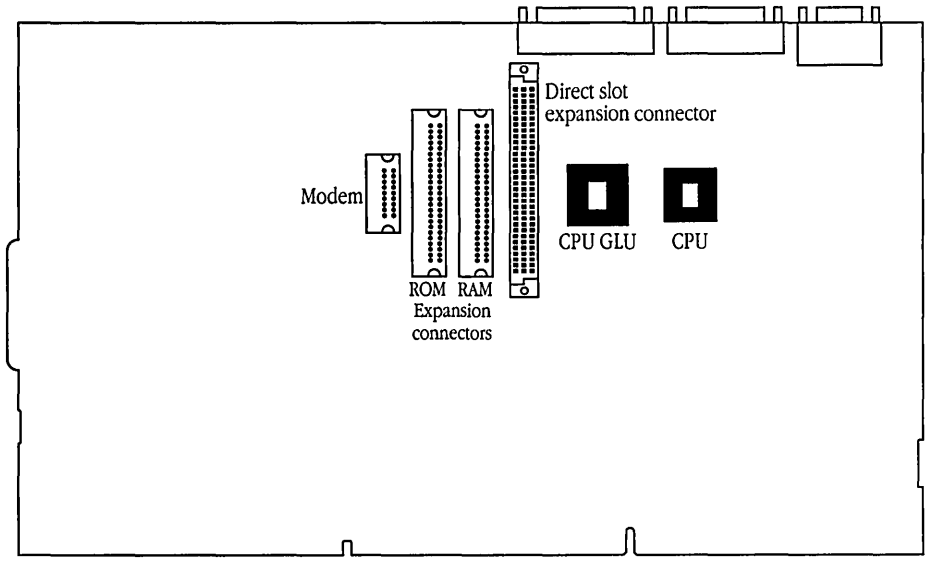
96 contact positions
2.54 mm (.100 inch) spacing sockets
Gold plated, 20 microinches, over nickel plate

Physical guidelines for Macintosh Portable expansion cards

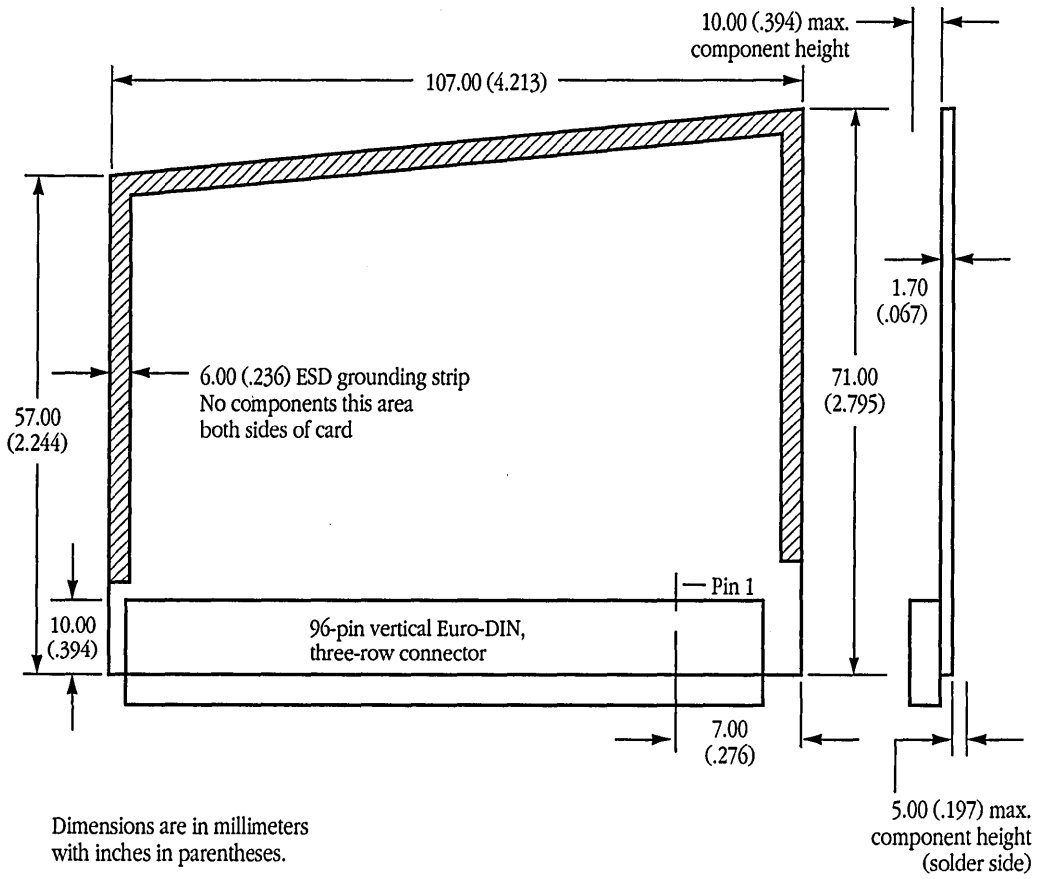
Figure 15-7 shows the location of the 96-pin expansion connector on the Macintosh Portable main logic board. Figure 15-8 is a design guide showing the size of the card, the location of the 96-pin connector, and the maximum allowable component mounting height. The Macintosh Portable uses the same Euro-DIN 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 15-4 and 15-6.

◆ *Note:* Before designing an expansion card for the Macintosh Portable, make sure you are aware of the limitations described in the section "The Macintosh Portable 68000 Direct Slot" in Chapter 13.

■ **Figure 15-7** Expansion connector location on Macintosh Portable main logic board



■ **Figure 15-8** The Macintosh Portable 68000 Direct Slot expansion card



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.

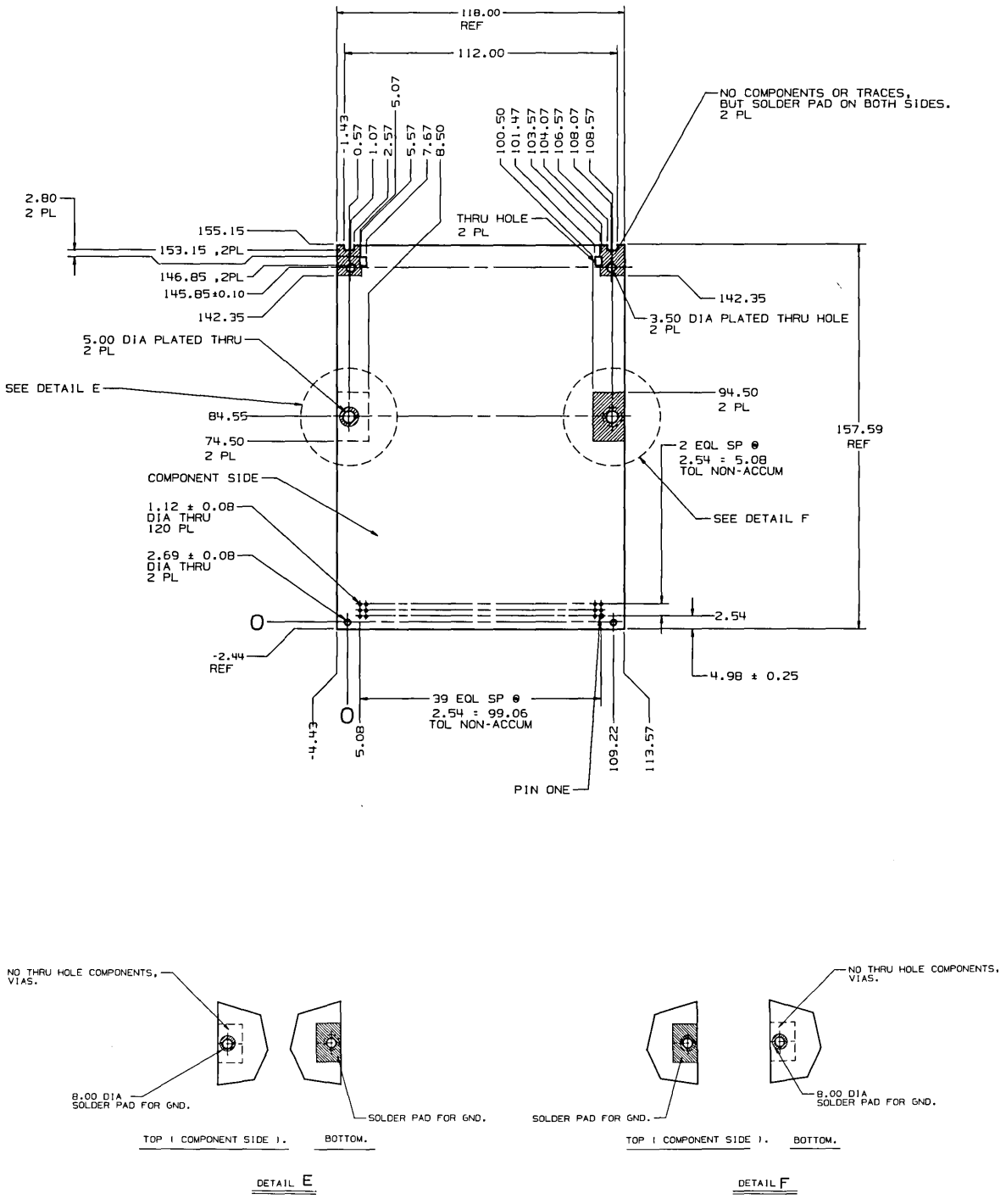
Figures 15-9 through 15-11 show design considerations for expansion cards that can be used in the Macintosh SE/30 and possibly in future 68030-based machines. Notice that you can design your card in either of two different sizes. Figure 15-9 shows the minimum allowable card size and Figure 15-10 shows the maximum allowable card size. Figure 15-11 shows the maximum component heights allowed on the two different card sizes.

Figure 15-12 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.

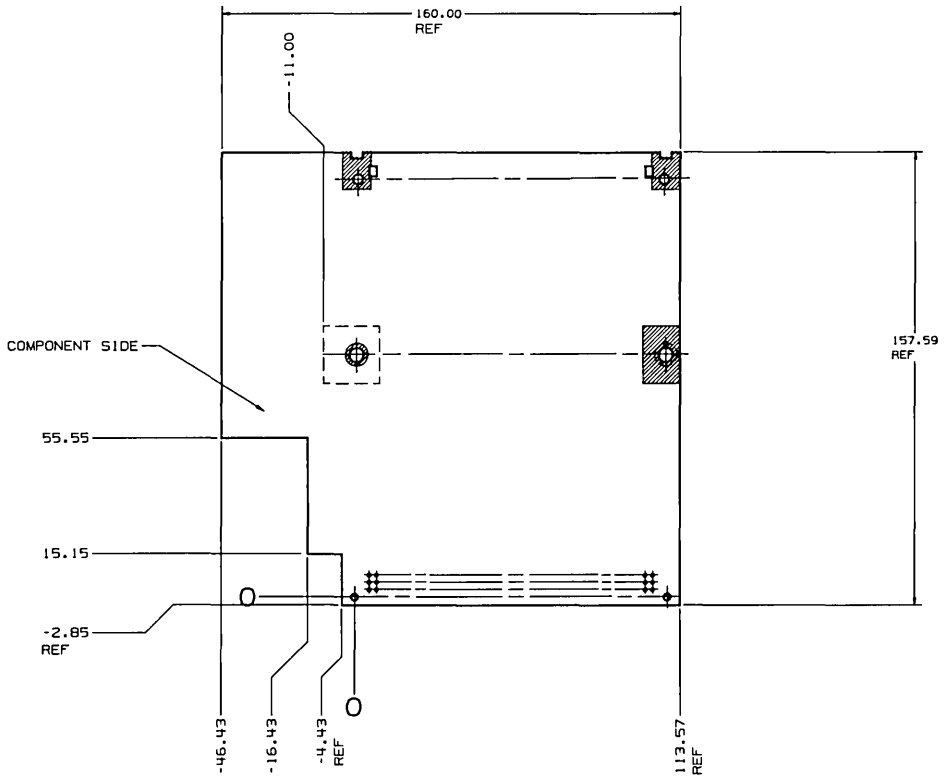
▲ **Warning** Figures 15-9 through 15-12 are from design guides used within Apple Computer. These drawings were correct at the time of publication but are subject to change in the future. ▲

Figure 15-13 shows how an expansion card mounts in the Macintosh SE/30 chassis. Figure 15-14 shows how the expansion card mounting clips should be oriented for two different revision levels of the main chassis.

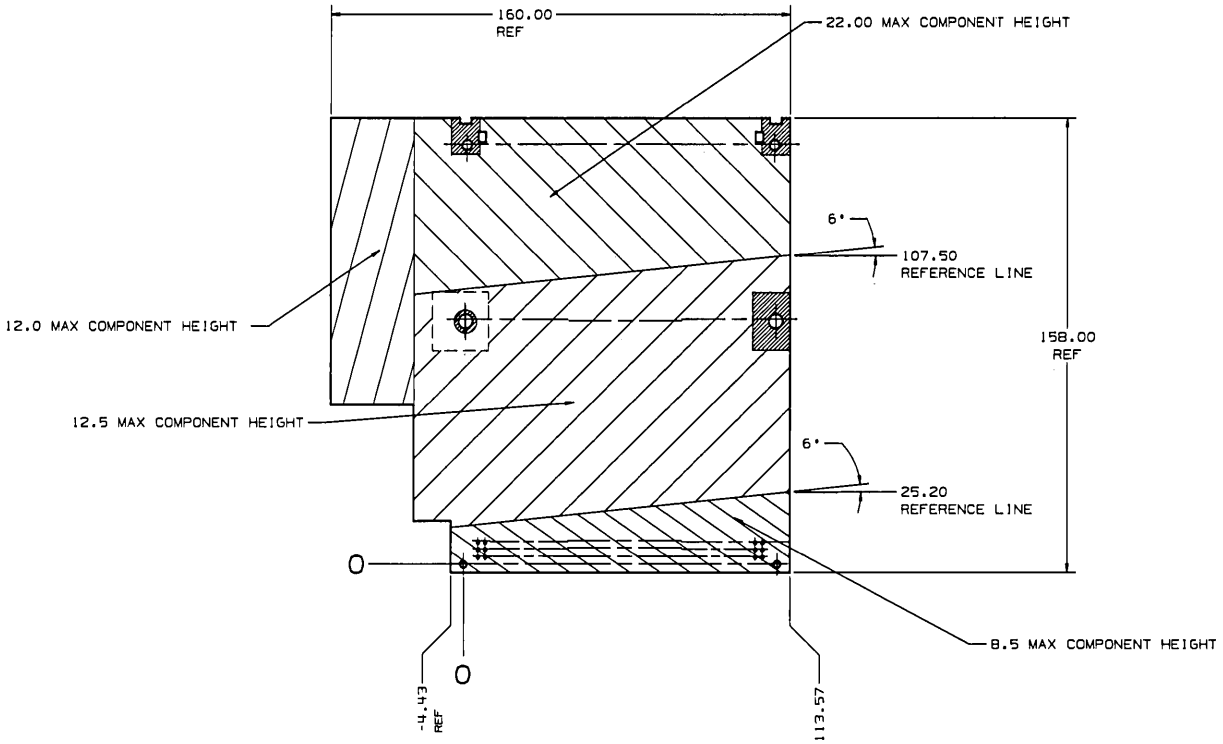
■ **Figure 15-9** Smallest allowable Macintosh SE/30 expansion card



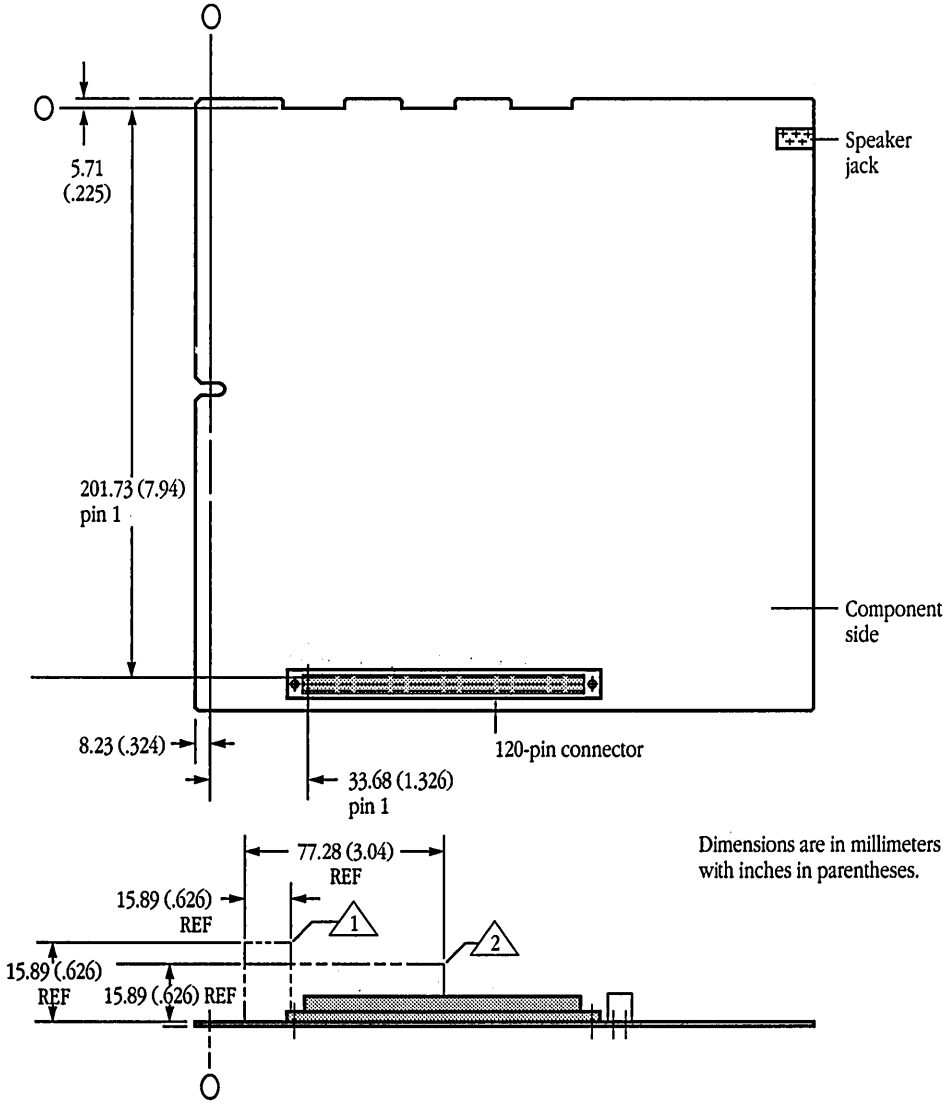
■ **Figure 15-10** Largest allowable Macintosh SE/30 expansion card



■ **Figure 15-11** Maximum allowable component heights for a Macintosh SE/30 expansion card



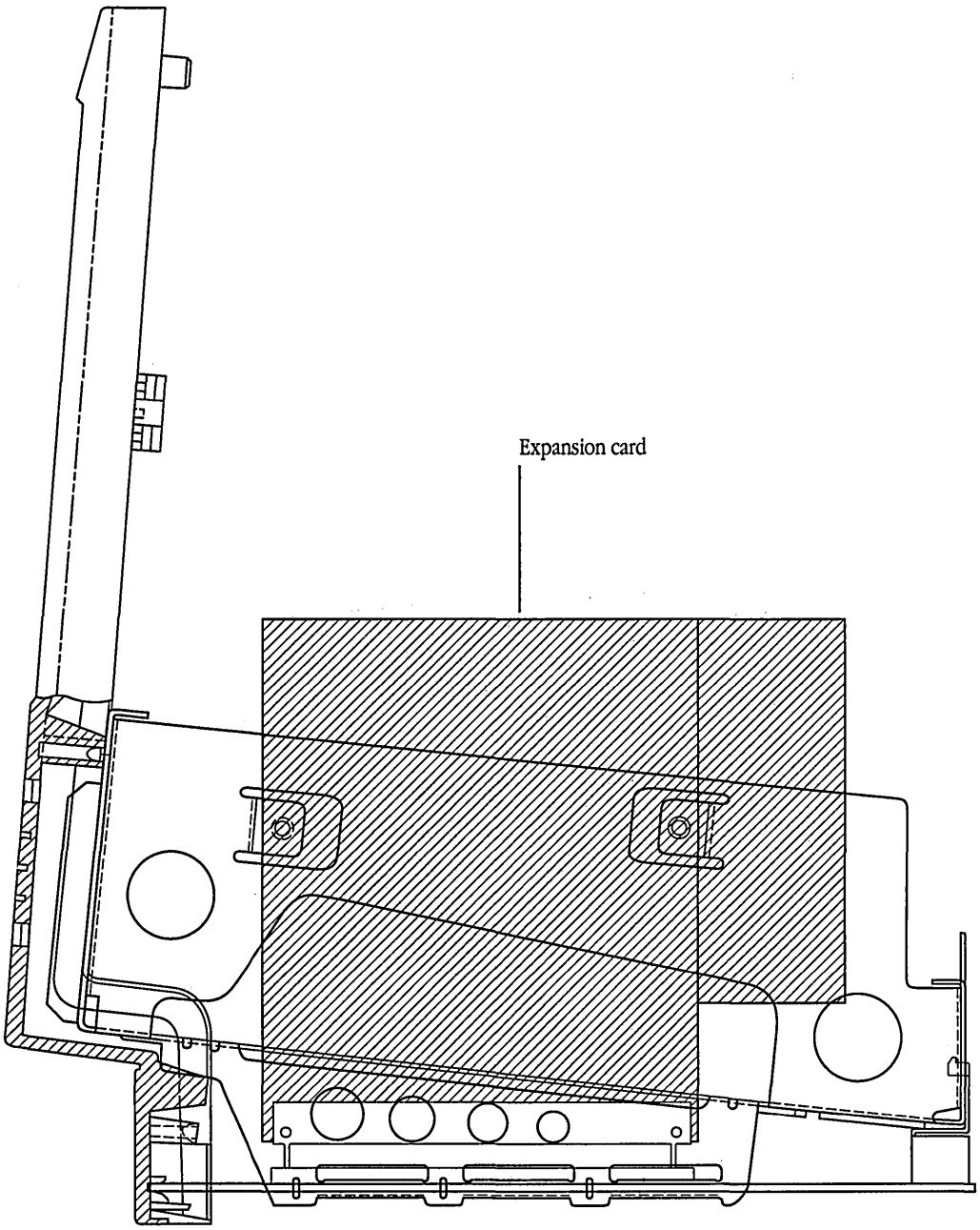
■ **Figure 15-12** Expansion connector on the Macintosh SE/30 main logic board



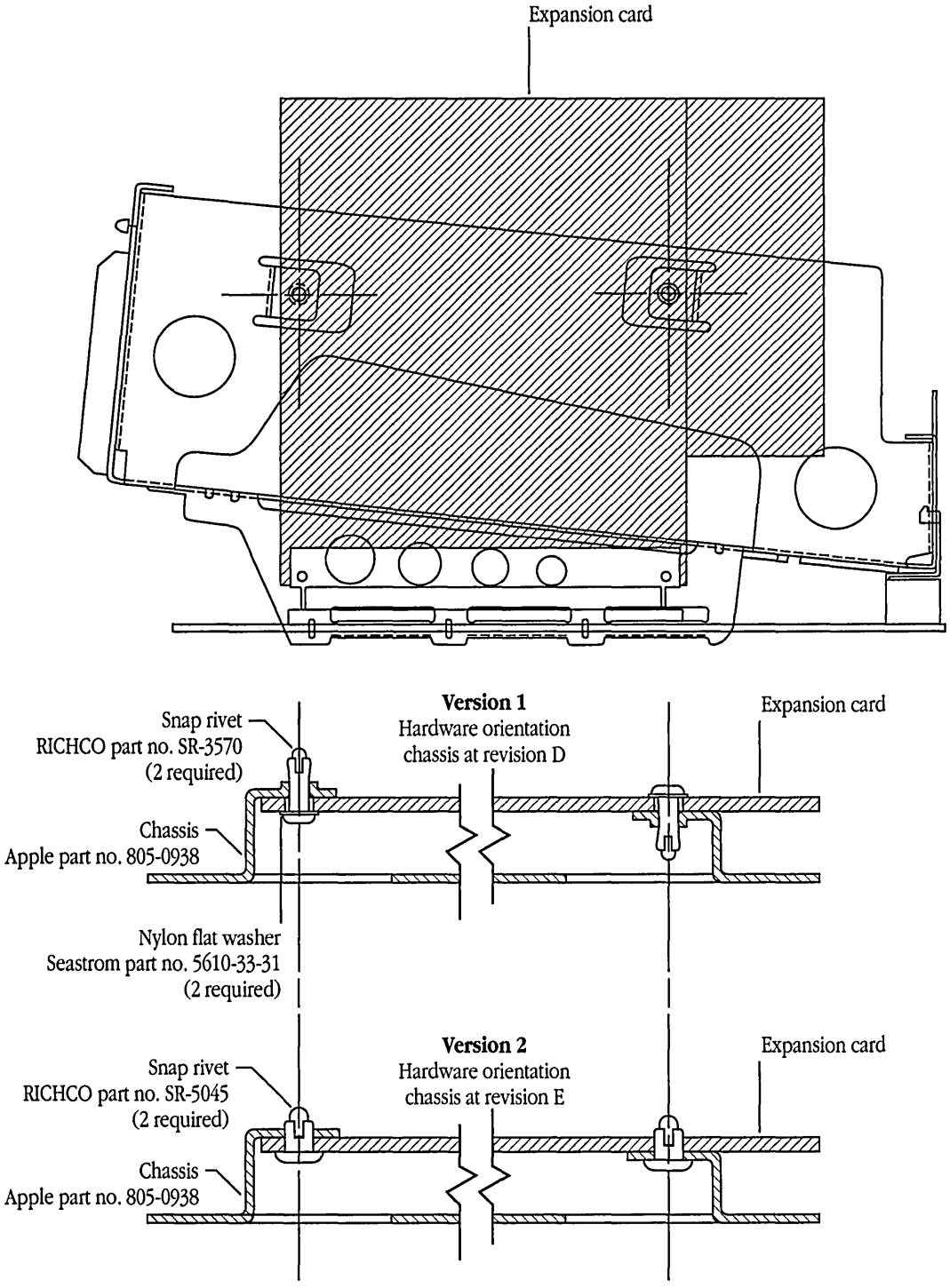
1 Indicated area represents space available for 31.50 (1.24) high (including socket) SIMMs module

2 Indicated area represents space available for 24.00 (.94) high (including socket) SIMMs module

■ **Figure 15-13** An expansion card in the Macintosh SE/30 assembly



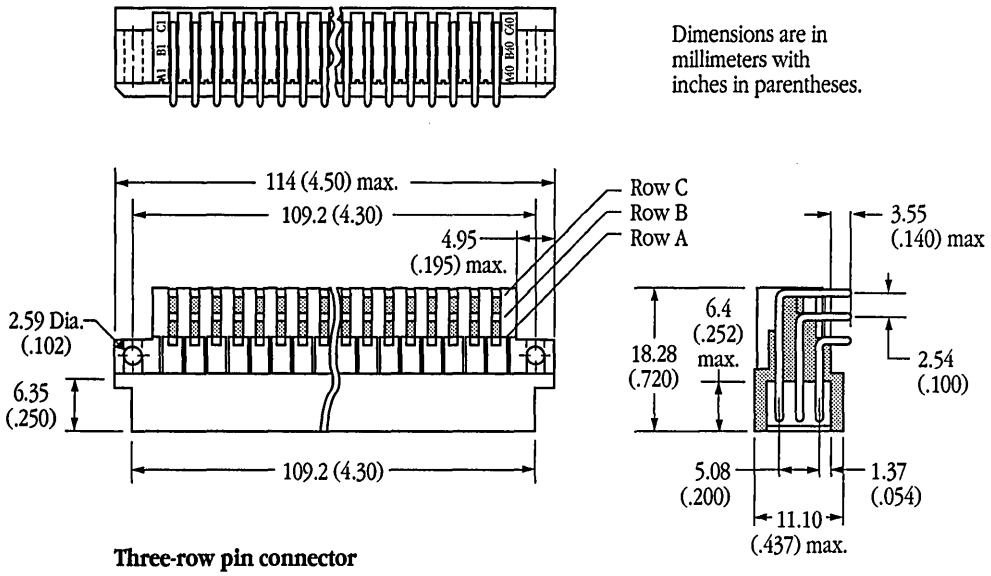
■ **Figure 15-14** Orientation of Macintosh SE/30 mounting hardware



The 68030 Direct Slot 120-pin connector for the Macintosh SE/30

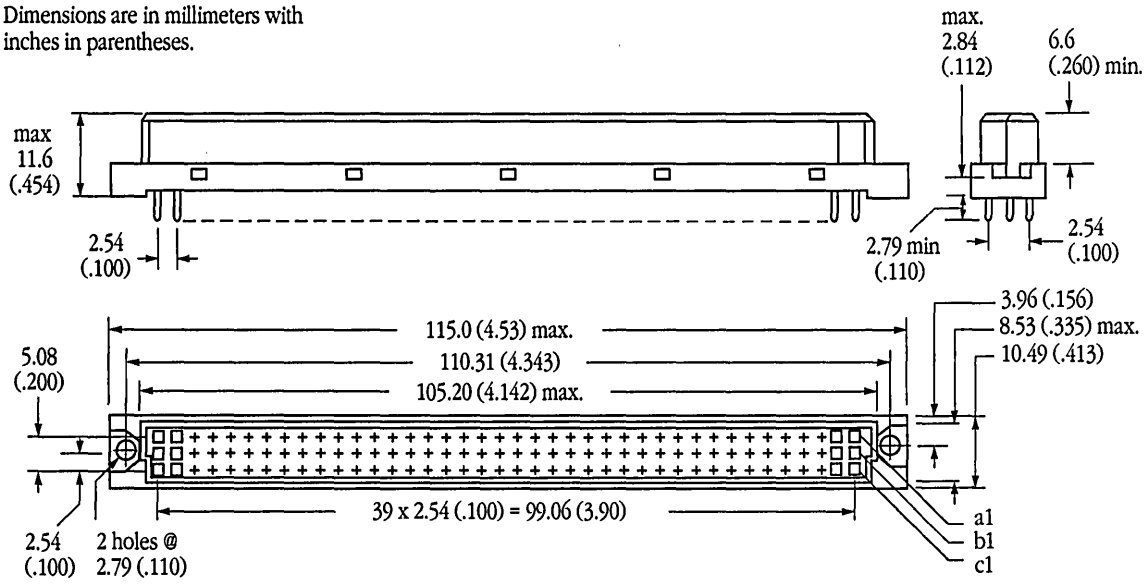
Figure 15-15 shows the plug connector that mates with the Euro-DIN 120-pin socket connector on the main logic board. Figure 15-12 shows the location of the 120-pin socket connector on the main logic board assembly. Figure 15-16 gives the prominent details of the socket connector.

■ **Figure 15-15** 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

■ **Figure 15-16** 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

Physical guidelines for Macintosh IIfx PDS expansion cards

The expansion card for the 68030 Direct Slot of a Macintosh IIfx computer is similar 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 6 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 as well as providing clearance dimensions for installing the card in the Macintosh IIfx computer.

▲ **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 future 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 15-15. 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 15-16.

External connection drawings

This section discusses both electrical and physical considerations required in making connections to external equipment.

The Macintosh SE and Macintosh SE/30 computers have 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 or the Macintosh SE/30.

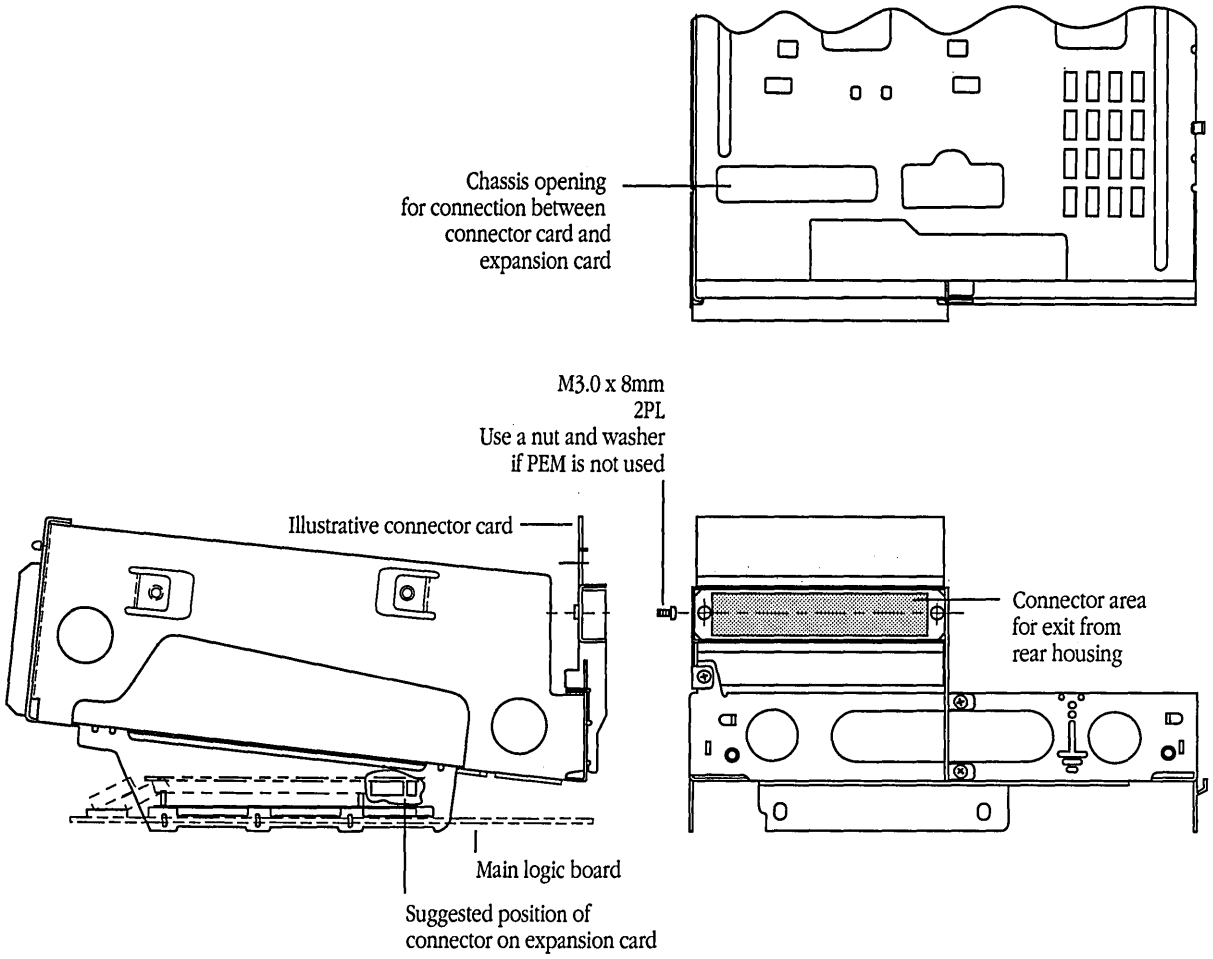
◆ *Note:* FCC regulations on radio-frequency emissions prohibit the installation of an external device access opening in the Macintosh Portable computer.

Figure 15-17 shows the Macintosh SE sheet metal and the structure for mounting your connector and connector card. With the exception of the horizontally mounted expansion card, this figure also applies to the Macintosh SE/30. Figures 15-18 and 15-19 show the recommended internal cable routing paths for the Macintosh SE and the Macintosh SE/30, respectively. Notice that the minimum allowable length for the internal cable on a Macintosh SE/30 is 220 millimeters (8.6 inches).

Foldout 7 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 7, it can be used on the Macintosh SE, the Macintosh SE/30, and future versions of the compact Macintosh.

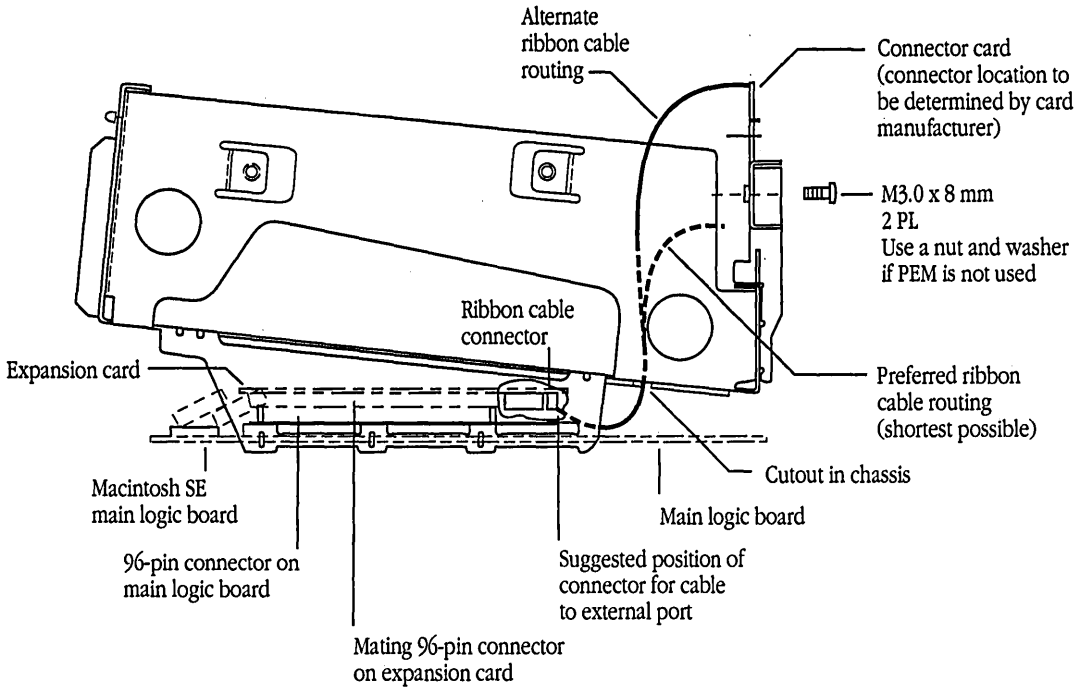
▲ **Warning** Foldout 7 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change in the future. ▲

■ **Figure 15-17** Connector card mounting on Macintosh SE chassis

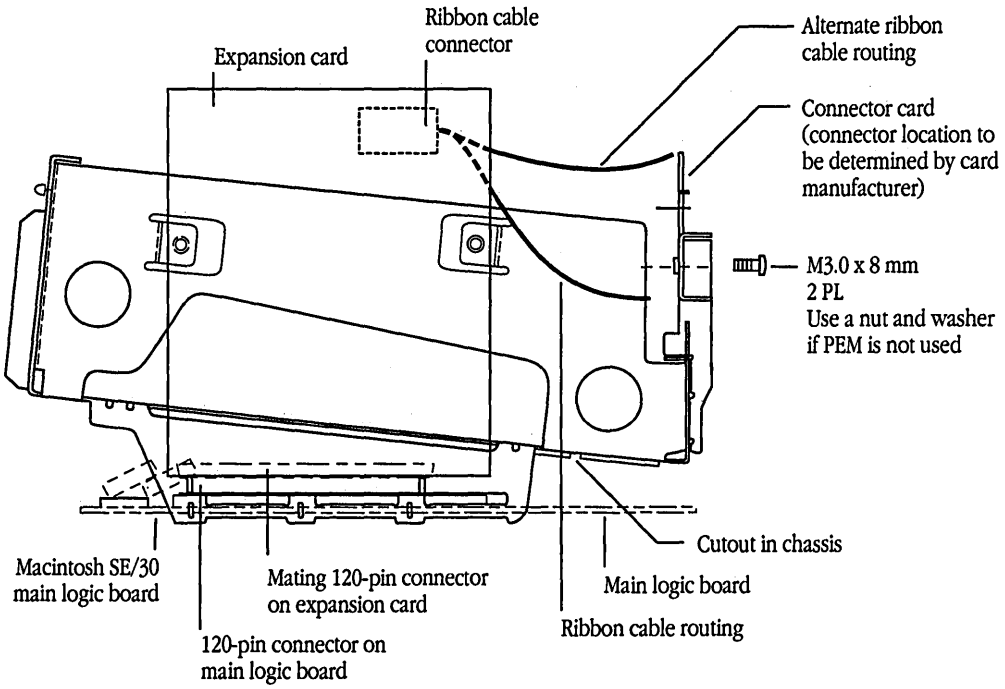


Except for the horizontally mounted expansion connector, the drawings in Figure 15-17 also apply to the Macintosh SE/30.

■ **Figure 15-18** Internal expansion cable routing for Macintosh SE



■ **Figure 15-19** Internal expansion cable routing for Macintosh SE/30



Note: The length of the ribbon cable must be 220 millimeters (8.6 inches) or longer between connectors.

Third-party design aids

A number of products are available from third parties that will make it easier and faster for you to design your expansion cards.

You can purchase blank Macintosh SE expansion cards for prototyping from

Diversified I/O, Inc.
1008 Stewart Drive
Sunnyvale, CA 94086
(408) 730-2171
AppleLink: D0242

Diversified I/O also has two other products that may be helpful in your expansion card design. One is the FreeBus Prototyping Kit, #451104, consisting of a board with a 96-pin connector that supports the Macintosh SE expansion slot. The other is a TwoBus Extender Kit, #450102, that extends the prototyping activities beyond the confines of the computer case and supports both Macintosh II and Macintosh SE extension.

You can obtain blank Macintosh SE/30 expansion cards for prototyping from Creative Solutions, Inc. Creative Solutions also provides a right angle extender that allows you to install your expansion card horizontally (component side facing up) outside of the chassis for debugging and testing. You can obtain more information or purchase these products by contacting

Creative Solutions, Inc.
4701 Randolph Rd., Suite 12
Rockville, MD 20852
(301) 984-0262

Chapter 16 **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) or 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 six-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 six-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 15-15, Figure 15-16, and Foldout 7 for drawings depicting the configuration.

Interface card block diagram

Figure 16-1 is a block diagram of the floppy disk controller. The controller card is made up of the following parts:

Control PALs: These 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 $80 000 through $8F FFFF.

Data bus transceivers: These 74LS245 buffers provide multiplexing control and sufficient current drive to and from the controller onto the data bus. During high-byte transfers, data is placed on D8–D15, while during low-byte transfers, the data goes on 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: This LSI chip contains the circuitry necessary to connect to the generic disk drive. Coupled with the companion disk interface IC chip, it handles all operations with the drive including read and write data, formatting, seeking, sensing drive status, and recalibrating.

Disk interface IC: This chip provides drive and timing support to the disk controller IC. It contains write precompensation and phase-locked loop circuitry.

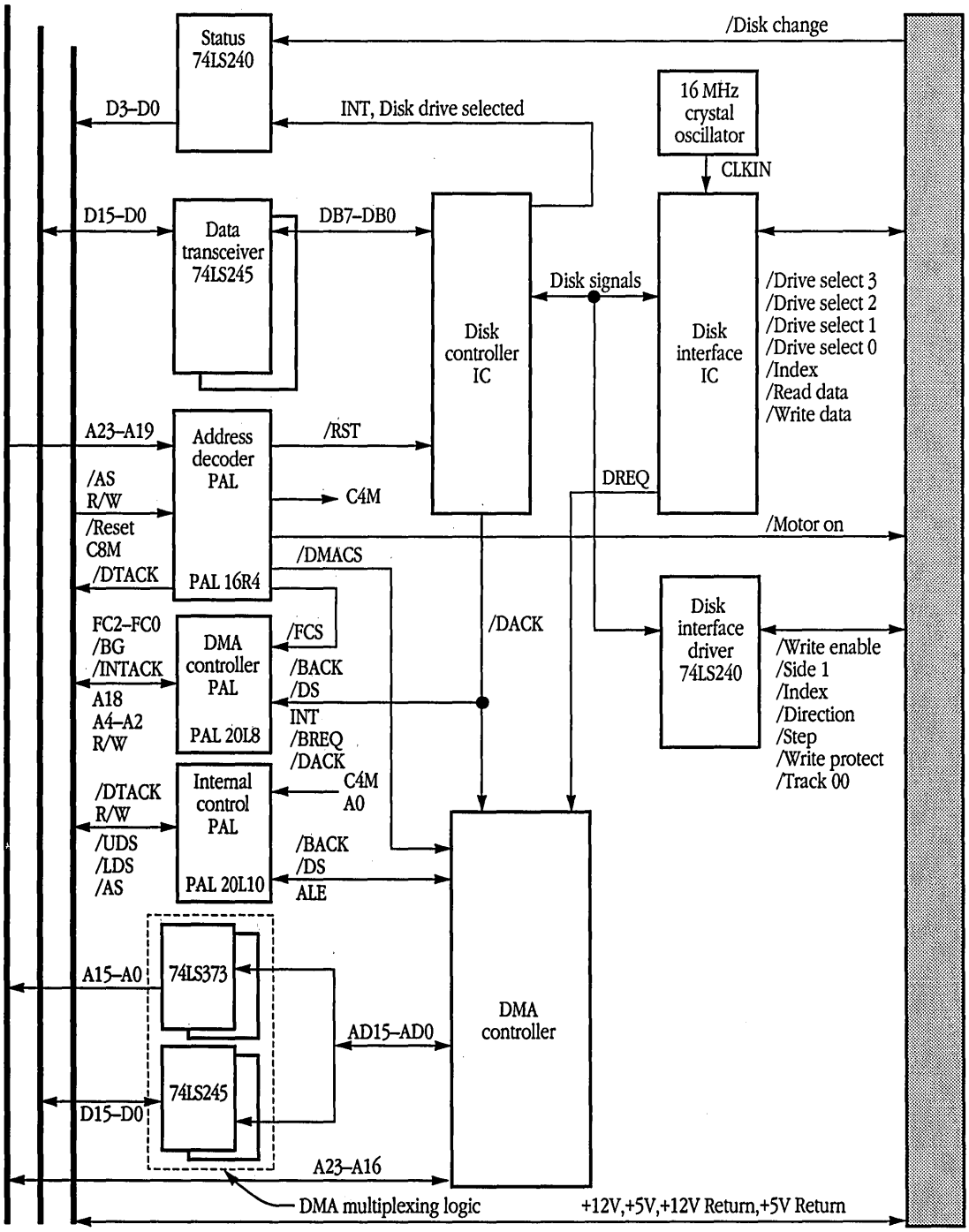
Disk interface driver: The disk interface driver buffers and provides current drive for several signals coming from and going to the disk drive. It also is used as a multiplexer for four of these signals.

16 MHz crystal clock oscillator: This oscillator provides a 16 MHz clock to the disk interface IC for use in the drive interface.

Dual-channel DMA controller and DMA control 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 16-1** 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 16-1.

■ **Table 16-1** Bus control signals

Signal name	Signal description
/AS	Indicates a valid address is on the address bus.
/UDS	Indicates that valid data is on the data bus D8–D15.
/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.
/DTACK	Signals that the data transfer cycle is completed.
/BR	Signals that the controller card would like to own the processor bus in order to perform a DMA transfer.
/BG	Signals to the controller that it owns the processor bus after completion of the current bus cycle.
FC0–FC2	These are the MC68000 processor status code and serve to signal an interrupt acknowledge cycle when they are all asserted high.
ALE	Signals that the DMA controller is gating a valid address onto the multiplexed address/data lines AD0–AD15.
/DS	Signals that data may be moved into or out of the DMA controller on the multiplexed AD0–AD15.
/DMACS	Selects the DMA controller during PIO transfers to or from it.

(Continued)

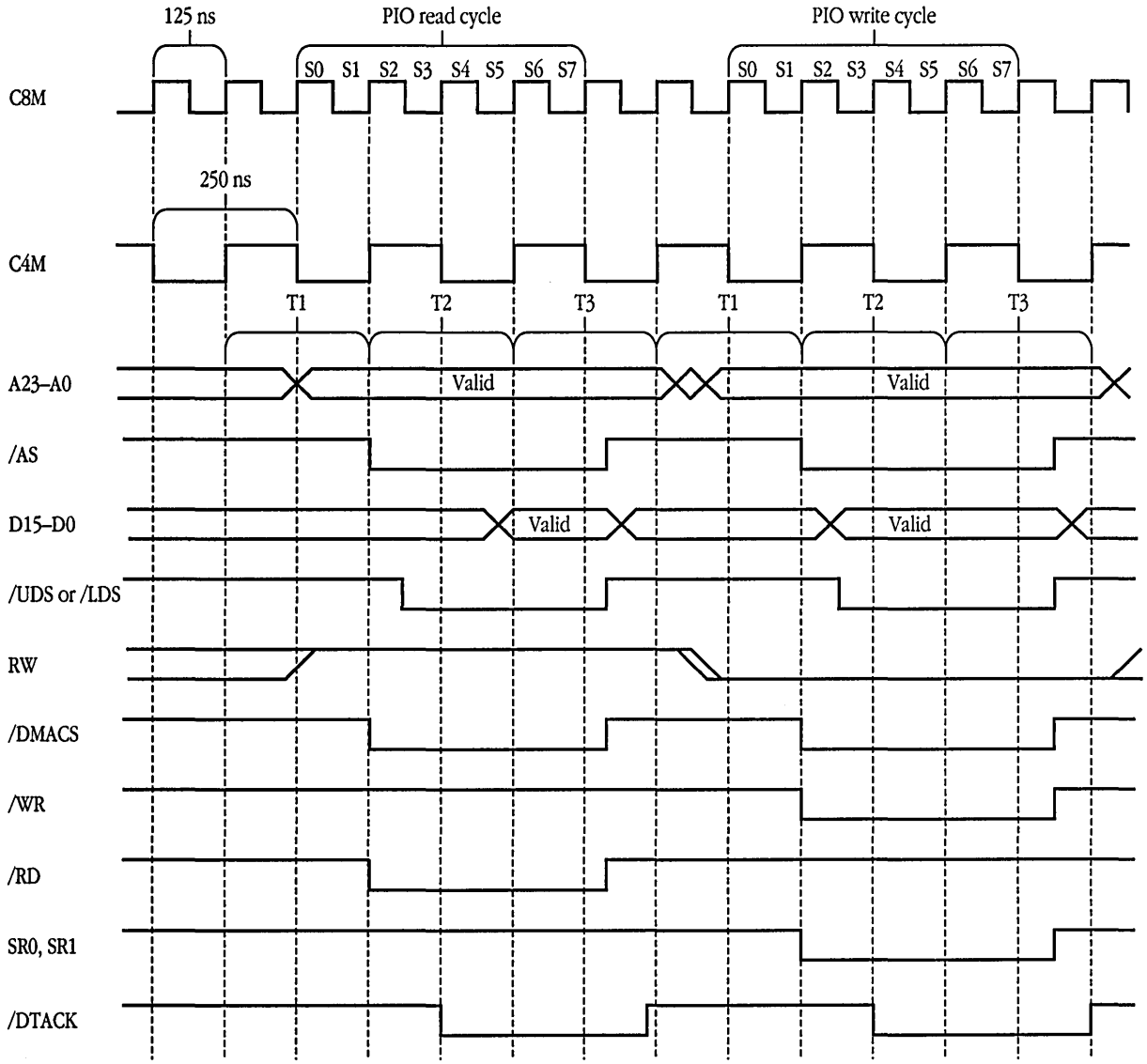
■ **Table 16-1** Bus control signals (Continued)

Signal name	Signal description
BREQ	Indicates that the DMA controller would like to take the processor bus.
BACK	Indicates that the DMA controller has the processor bus.
/DREQ	The disk interface IC makes a DMA request to the DMA controller.
/DACK	The DMA controller acknowledges the disk interface IC's DMA request.
/EOP	Signals that a disk read or write command has been terminated because the data requested has been transferred.

Programmed I/O (PIO) 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 /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 16-2 for signal timing.

■ **Figure 16-2** 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 16-1. 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:* /PMCYC is the signal 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. See Figure 13-2 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 zero (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 16-2.

■ **Table 16-2** Device select decode addresses

Decode address range	Device selected and action resulting
$80 0000–$83 FFFF	Read from main status register of disk controller IC. A write to this address turns RST on (resets the disk controller IC). Also, read additional status register. The main status register is on the least significant byte and the additional status register is on the most significant.
$84 0000–$87 FFFF	Read or write control, status, and data information to the data register in the disk controller IC.
$88 0000–$8B FFFF	Write turns drive motor to on, read turns motor and controller's reset signal off. (Interrupts are enabled when the motor is on!)
$8C 0000–$8F FFFF	Read from or write 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 two chapters. Chapter 17 provides the electrical and mechanical information you need to design ROM, RAM, and modem expansion cards for the Macintosh Portable computer. Included are expansion connector pinouts and signal descriptions, address space allocations, physical design guides, and hardware and software design suggestions.

Chapter 18 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 17 Macintosh Portable RAM, ROM, and Modem Expansion

In addition to the processor-direct slot expansion interface described in Chapters 13 through 15, the Macintosh Portable main logic board includes three expansion connectors, two for add-on memory expansion cards and one for a modem card. One memory connector is for ROM expansion and the other is for RAM expansion. This chapter provides the electrical and mechanical details you need to design ROM expansion, RAM expansion, and modem cards for 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 electronic disks (EDisks).

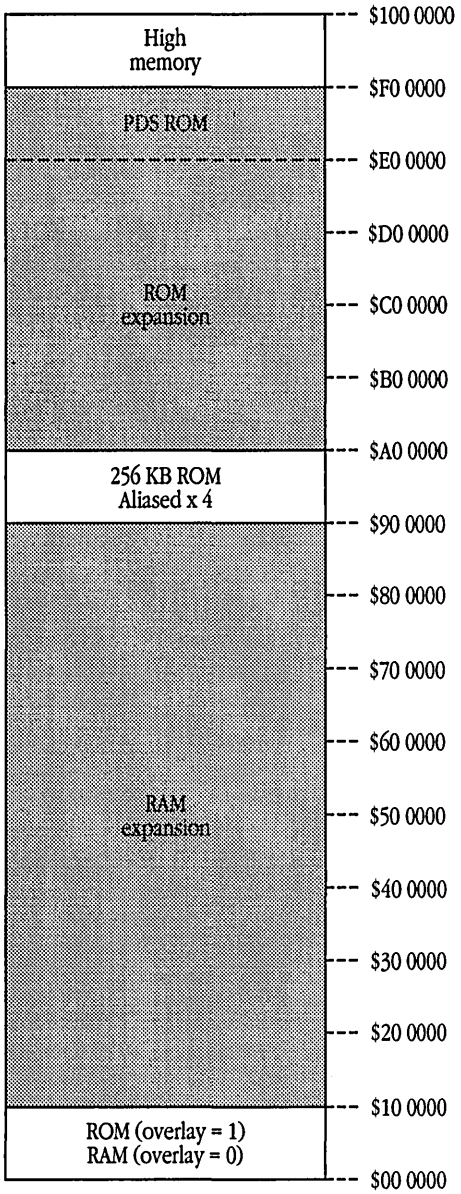
ROM expansion address space

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 ROM space shown in the memory map of Figure 17-1.

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. △

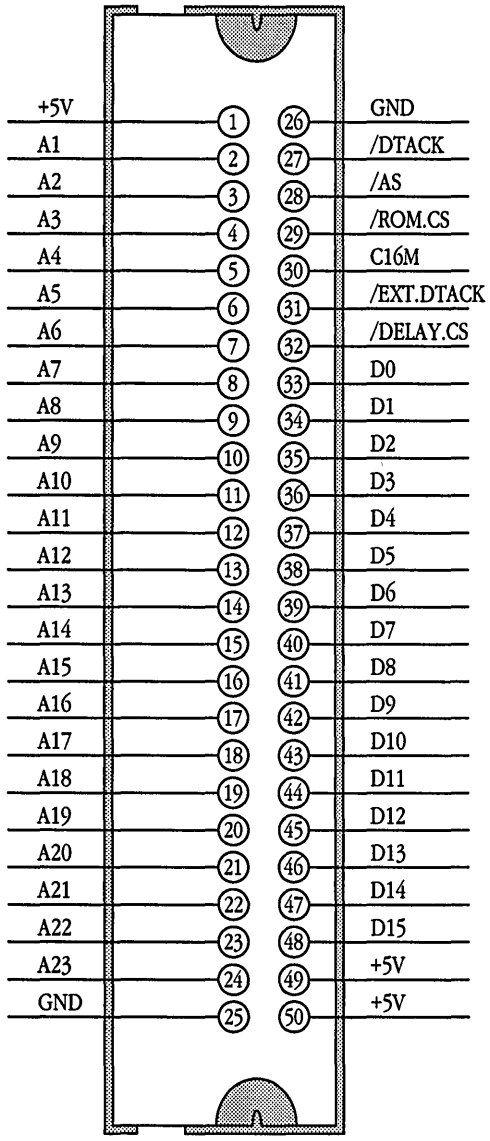
■ **Figure 17-1** Macintosh Portable memory map



ROM expansion cards

Your ROM expansion card connects to the Macintosh Portable through a single 50-pin connector (slot) on the Macintosh Portable main logic board. Refer to Figure 15-7 for the location of the connector on the main logic board. Figure 17-2 shows the pinout for the ROM expansion connector.

■ **Figure 17-2** 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 17-1 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.

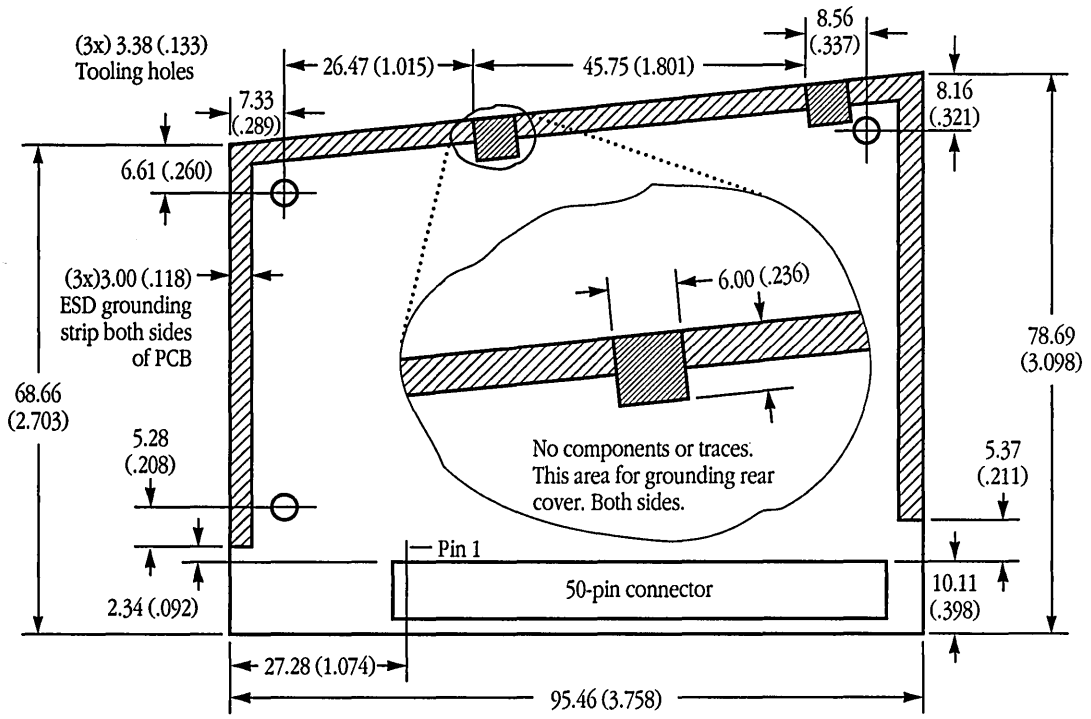
■ **Table 17-1** Macintosh Portable ROM expansion connector signals

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	External data transfer acknowledge signal that disables main system /DTACK signal
32	/DELAY.CS	The CPU GLU chip generates this signal 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 17-3 is a design guide providing the physical specifications you need to design a ROM expansion card for the Macintosh Portable.

▲ **Warning** Figure 17-3 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change in the future. ▲

■ **Figure 17-3** ROM expansion card design guide



Dimensions are in millimeters
with inches in parentheses.

Design considerations and suggestions

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 x 8 bit or 512K x 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 will 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 however, the amount of space and the number of ROM chips on a ROM expansion card are limited. When designing your ROM expansion card, 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 requiring an expensive upgrade.

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 that record data on rotating magnetically coated disks.

- *Note:* Third-party developers are currently limited to developing ROM EDisks only, 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. The ROM EDisks behave just like an internal RAM EDisk except that they are read only and cannot be resized.

The EDisk driver

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 the Disk Driver chapter of *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 zero, but is rotated left by one bit before each longword is added in. The following assembly code example demonstrates the algorithm.

```
        Lea             The Block,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 to 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

This is a 128-byte field that 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

This 2-byte field indicates the size of the EDisk header block. The size is currently 512 bytes.

HdrVersion

This 2-byte field indicates the version of the EDisk. The version number is currently $0001.

HdrSignature

This 12-byte 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

This 4-byte field indicates the size of the device in bytes, which 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

This 4-byte field indicates the time of day when the EDisk was last formatted. The EDisk driver updates this field for RAM-based EDisks when the Format control call is made. This information may be useful in uniquely identifying a RAM-based EDisk.

HdrFormatTicks

This 4-byte 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 the Format control call is made. This information may also be useful in uniquely identifying a RAM-based EDisk.

HdrCheckSumOff

This 4-byte field is the offset (from the beginning of the header block) of the checksum table, or zero if checksumming should not be performed on the EDisk.

HdrDataStartOff

This 4-byte field is the offset (from the beginning of the header block) of the first block of EDisk data.

HdrDataEndOff

This 4-byte 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

This 4-byte 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 zero indicates that the EDisk driver should use the default media icon for this EDisk.

HdrDriveIconOff

This 4-byte 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 zero indicates that the EDisk driver should use the default drive icon for this EDisk.

HdrWhereStrOff

This 4-byte field is the offset (from the beginning of the header block) of the Pascal string that describes the disk location for the Finder Get Info command. An offset of zero indicates that the EDisk driver should use the default string for this EDisk.

HdrDrive Info

This 4-byte field should be returned by the Drive Info control call. A value of zero indicates that the EDisk driver should use the default drive info for this EDisk.

Macintosh Portable RAM expansion

The RAM 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.

This section describes the RAM expansion address space, and gives the electrical and mechanical information you need to design a RAM expansion card.

RAM expansion address space

The 1 MB permanent RAM memory is arranged as a 512K x 16 bit array. This RAM array is located between addresses $00 0000 and $0F FFFF in the Macintosh Portable memory map (Figure 17-1), and is overlaid by the system ROM after a system reset and before the first ROM access.

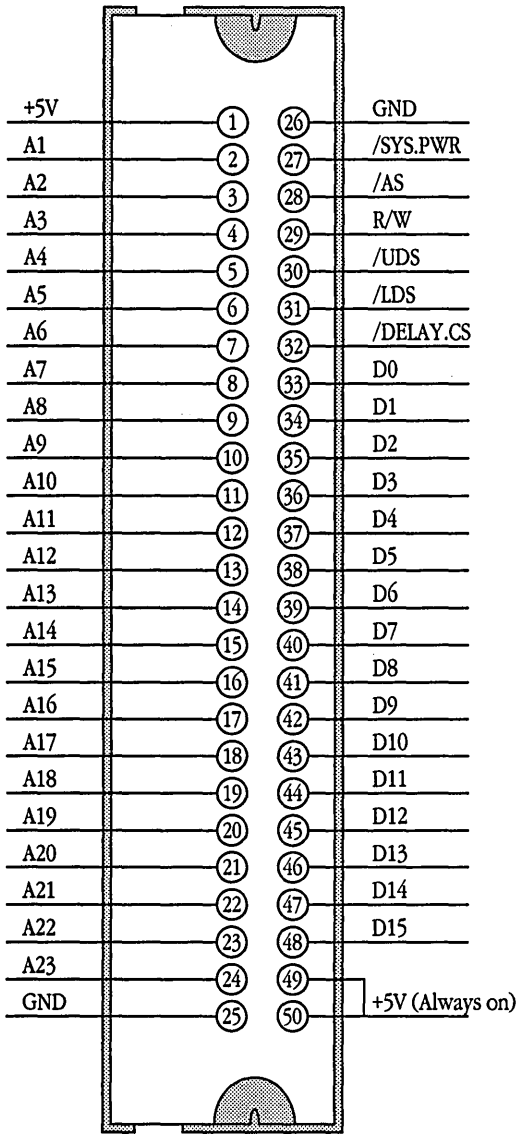
The 8 MB space between addresses $10 0000 and $8F FFFF is reserved for RAM expansion. Theoretically you could design an 8 MB RAM expansion card, but the RAM expansion connector provides addressing for only 4 MB. You can design your expansion card for any of a number of possible configurations of additional RAM. For example, Apple has designed a 1 MB RAM expansion card. The 1 MB expansion card is arranged as a 512K x 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 1.5M x 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.

RAM expansion cards

Your RAM expansion card connects to the Macintosh Portable through a single 50-pin connector (slot) on the Macintosh Portable main logic board. Refer to Figure 15-7 for the location of the connector on the main logic board. Figure 17-4 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 17-2 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 17-4** Macintosh Portable RAM expansion connector pinout



■ **Table 17-2** 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	Permanent ROM /CS signal
30	/UDS	16 MHz system clock
31	/LDS	External /DTACK signal that is an input to the CPU GLU chip
32	/DELAY.CS	The CPU GLU chip generates this signal 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

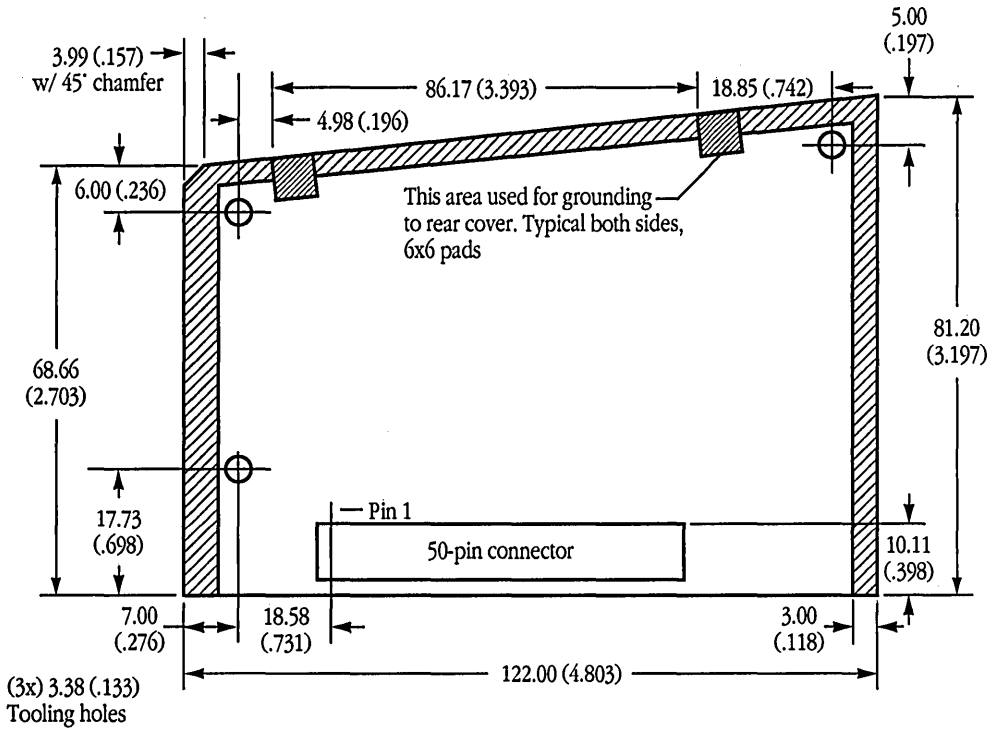
There is one 68HC000 processor wait state when accessing memory locations in the expansion RAM. This access requires a bus cycle of nominally 320 ns. Like permanent RAM, there is no device contention for bandwidth other than the 68HC000 processor, and because the memory array is static RAM, it does not 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 17-5 is a design guide providing the physical specifications you need to design a RAM expansion card for the Macintosh Portable.

▲ **Warning** Figure 17-5 is from a design guide used within Apple Computer. This drawing was correct at the time of publication but is subject to change in the future. ▲

■ **Figure 17-5** RAM expansion card design guide



Dimensions are in millimeters
with inches in parentheses.

Macintosh Portable modem card

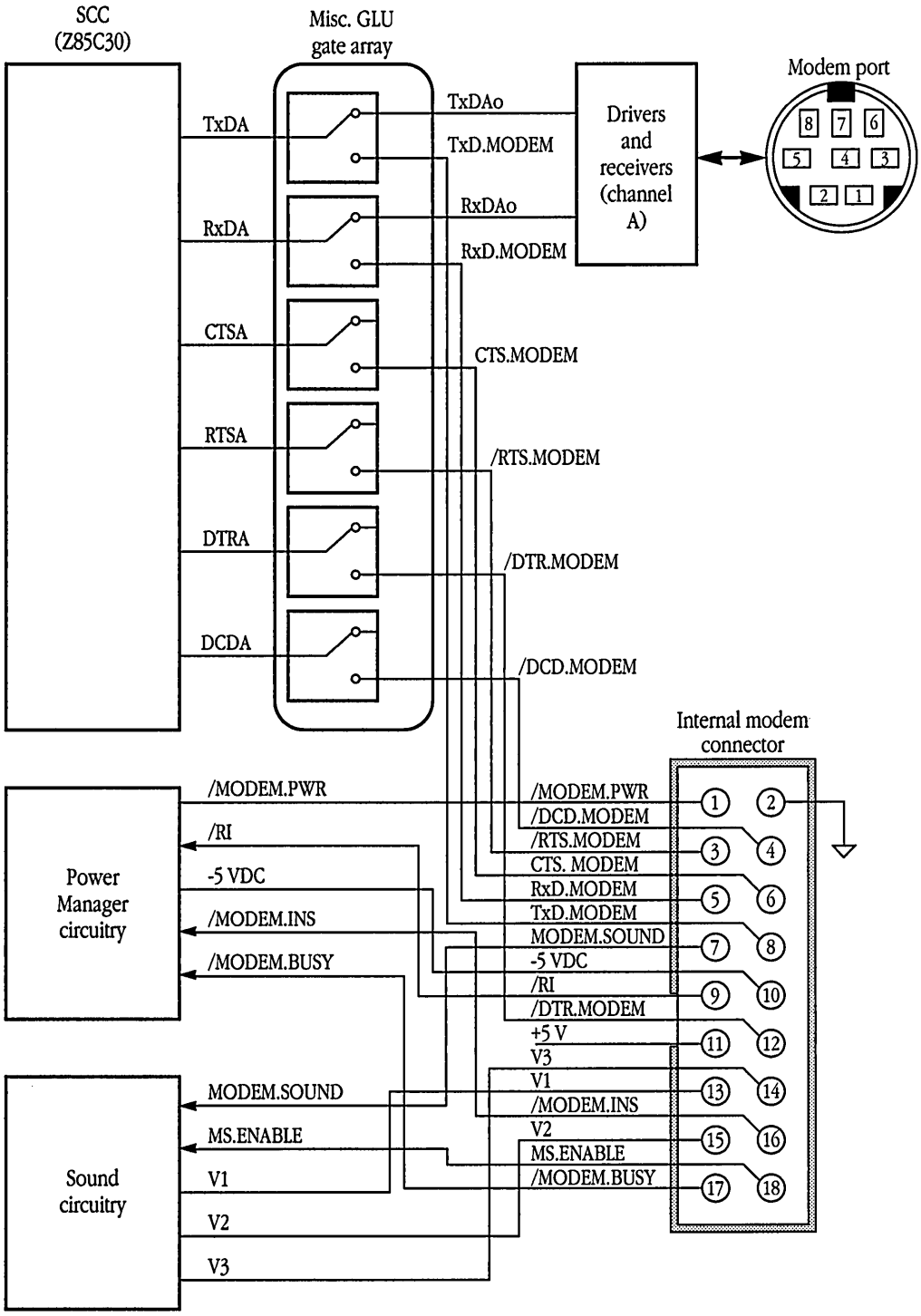
The main logic board of the Macintosh Portable includes an 18-pin, internal modem connector. The connector accommodates an Apple modem card or a compatible third-party modem card.

This section provides information you need if you are developing your own modem card and software.

Modem card hardware interface

Figure 17-6 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.

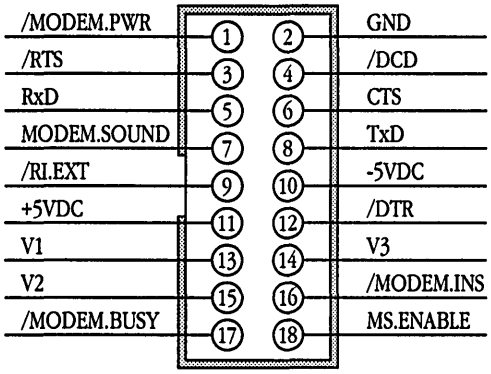
■ **Figure 17-6** Macintosh Portable modem interface



Modem connector electrical interface

The modem card connects to the Macintosh Portable through an 18-pin dual inline socket connector (slot). The data is at CMOS levels (that is, V_{IL} = 0 to 0.8V; V_{IH} = 3.5 to V+; I_{OL} = 1.6mA; I_{OH} = – 25µA). Refer to Figure 15-7 for the location of the connector on the main logic board. Figure 17-7 shows the pinout of the modem connector. Table 17-3 provides the name and description of each signal available at the modem connector.

■ **Figure 17-7** Pinout of modem connector on Macintosh Portable



■ **Table 17-3** Modem connector signal descriptions

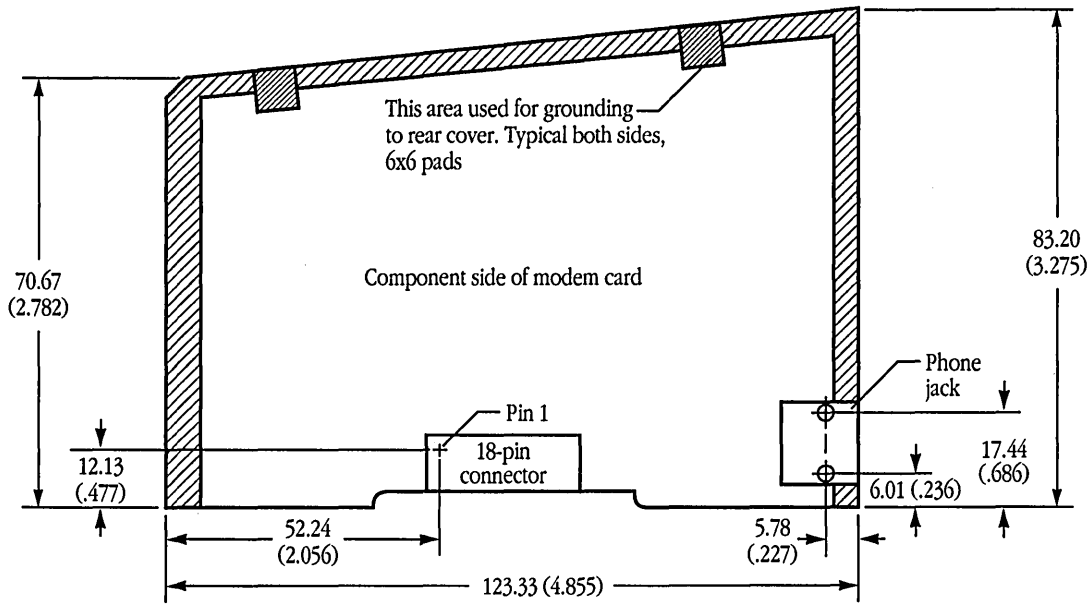
Pin number	Signal name	Signal direction	Signal description
1	/MODEM.PWR	Output	Active low signal from the power manager; see the section "Modem Power-Control Interface," later in this chapter.
2	GND	—	Ground.
3	/RTS	Output	Request to Send signal from the computer to the modem.
4	/DCD	Input	Data Carrier Detect; the behavior of the Data Carrier Detect signal depends on the state of the &C command.
5	RxD	Input	Data Received; connected to the RxD pin on the SCC.
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	–5 VDC*	—	–5V power; the –5V 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	+5 VDC	—	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 modem card

Figure 17-8 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 17-8** Modem card design guide



Dimensions are in millimeters with inches in parentheses.

Modem power-control interface

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)

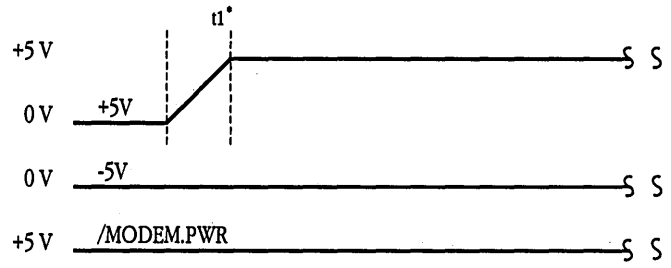
If the modem is executing any self-tests, it is considered to be executing a command and therefore busy.

The Macintosh Portable includes a Power Manager IC that controls the /MODEM.PWR signal. See Figures 17-9 and 17-10 for timing diagrams of the cold-start and warm-start 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 Apple, 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 high (/DCD and RxD) and one of its outputs low (MS.ENABLE).

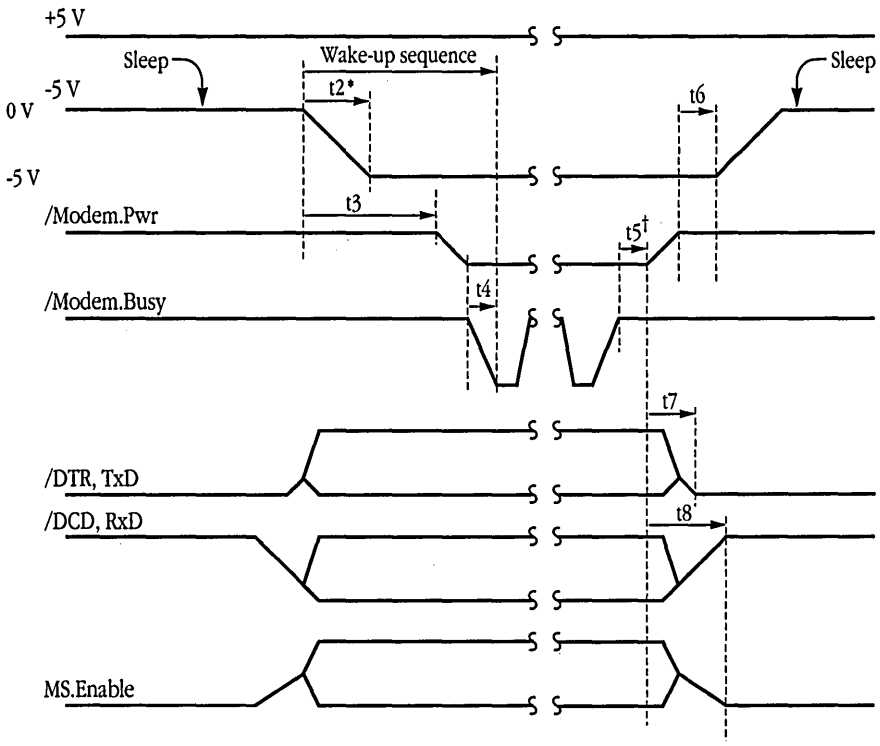
Usually, the power manager does not negate /MODEM.PWR if the modem has /MODEM.BUSY asserted. However, there are times when the Power Manager IC must turn the modem off even though it is busy; for example, when the battery reserve is too low. If this occurs, the modem must stop its activity (for example, 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 17-9** Cold-start (initial power-up) timing diagram



* After t1, maximum overshoot is within 50 mV peak to peak.

■ **Figure 17-10** Warm-start (wake-up) timing diagram



	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 signal (/RI.EXT) 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 eight-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 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 Hz ± 20 Hz @ 6 ±1 dB below the transmit carrier level
550 Hz ± 20 Hz @ 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 18 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.

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 shown in Figures 15-15 and 15-16, 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 the Macintosh SE/30 and Macintosh IIfx computers.

▲ **Warning** Cards designed for the Macintosh IIci cache connector are not compatible with cards designed for the 68030 Direct Slot on the Macintosh SE/30 or the Macintosh IIfx computers. Their pinouts, form factors, clock speeds, and power budgets are different. Any attempts 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 the 68030 Direct Slot; 5 watts of power is allocated at +5 volts, and +12 volts is not available.
- The Macintosh IIci case does not have sufficient space for an external device access opening. This prevents you from installing 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, 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 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 frequently 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 if 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 to $52FF FFFF in the Macintosh IIci memory map is reserved for cache memory. Table 18-1 shows the 8 MB address spaces that are reserved for the cache data and tag memories.

■ **Table 18-1** 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 18-2.

■ **Table 18-2** 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 18-1 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.

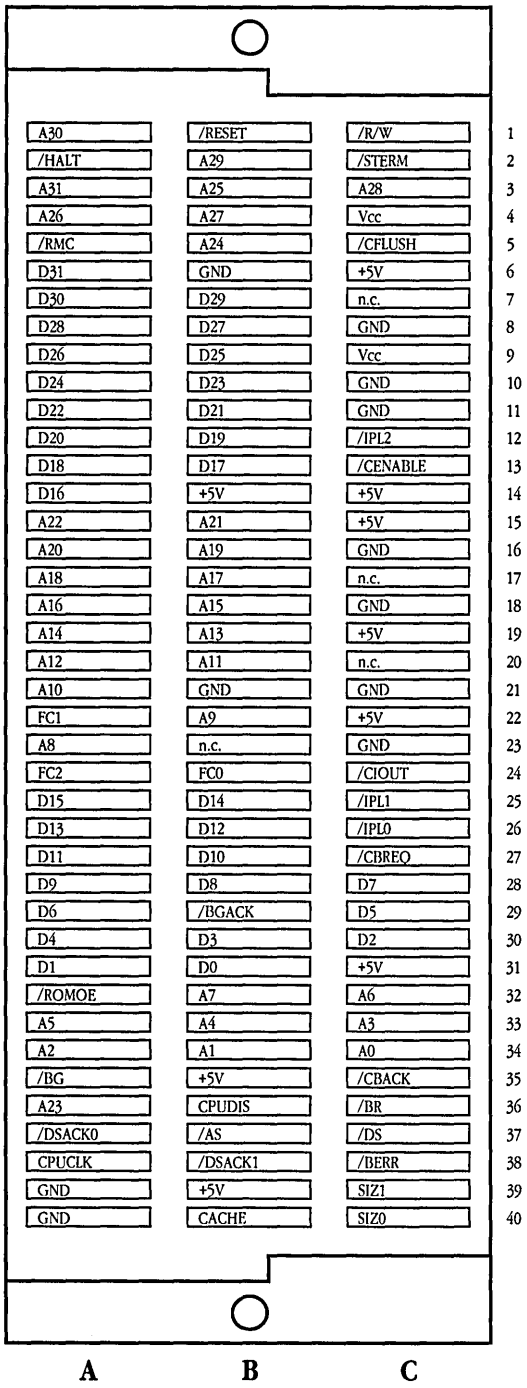
The diagnostic signals (/ROMOE, /DSACK0–/DSACK1, /IPL0–/IPL2, /BR, and CPUDIS) are needed for Apple's internal use in debugging and for emulator support. They are documented so that third-party developers can easily make use of emulators or other hardware debugging tools. The diagnostic signals are not required for cache memory operations and may not be supported in future implementations of the cache connector.

Table 18-3 lists the cache connector signal names and gives a brief description of each signal. Table 18-4 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. The special-purpose diagnostic signals are shown in **boldface** type in both tables.

In the column labeled *Input/output* in Table 18-4, *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 18-4, 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 pullup" in the table means that the signal is driven low, and a 1 kilohm pullup resistor on the main logic board returns the line to a logic 1.

Correspondingly, /BERR presents a drive of 40 µA/.4mA, 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 .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.

■ **Figure 18-1** Macintosh IIci cache connector pinout



■ **Table 18-3** 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
/RESET	System reset
/BERR	Bus error
/HALT	Halt
FC0–FC2	Function codes, bits 0 through 2
/BR	Bus request
/BG	Bus grant
/BGACK	Bus grant acknowledge
SIZ0–SIZ1	Transfer size, bits 0 and 1
/AS	Address strobe
/DSACK0 –/DSACK1	Data transfer and size acknowledge, bits 0 and 1
R/W	Read/write
/STERM	Synchronous termination
/CBACK	Cache burst acknowledge
/CBREQ	Cache burst request
/CIOUT	Cache inhibit out
/DS	Data strobe
/RMC	Read-modify-write cycle
/IPL0 –/IPL2	Interrupt priority lines, bits 0 through 2
CPUCLK	25 MHz CPU clock
/ROMOE	ROM output enable
CPUDIS	CPU disable
CACHE	Memory controller for cache access
CFLUSH	Cache flush
CENABLE	Cache enable
n.c.	No connection
+5V	+5 volts
GND	Ground

In Table 18-4, where a signal is shown in parentheses, it is usually an output that is driven by the MC68030 but is tri-stated by the processor after responding to a bus request. When tri-stated by the MC68030, this signal may be driven by the cache expansion card.

In addition to the special diagnostic signals, some of the functions shown in Table 18-4 are used only during diagnostic testing. An asterisk following an input or output designation or a load or drive parameter indicates 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 an asterisk. 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 18-4** 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/.4 mA, 30 pF
A30–A31	(In) */Out	(Load: 300 µA/8 mA, 100 pF) * Drive: 40 µA/.4 mA, 30 pF 1 kΩ pullup
D0–D23	In/Out	Load: 150 µA/1 mA, 100 pF Drive: 40 µA/.4 mA, 30 pF
D24–D31	In/Out	Load: 300 µA/1 mA, 100 pF Drive: 20 µA/.2 mA, 30 pF
/RESET	In */Out	Load: na/15 mA, 50 pF * Drive: 20 µA/.2 mA, 15 pF Open collector, 470 Ω pullup
/BERR	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF 1 kΩ pullup
/HALT	In/Out	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF 1 kΩ pullup
FC0–FC2	(In)*/Out	(Load: 100 µA/8 mA, 50 pF) * Drive: 20 µA/.2 mA, 15 pF 1 kΩ pullup

(Continued)

■ **Table 18-4** Macintosh IIci cache connector signals, loading or driving limits (Continued)

Signal name	Input/output	Load or drive limits
/BR	In *	Drive: 40 µA/.4 mA, 30 pF * 1 kΩ pullup *
/BG	In */Out	(Load: 100 µA/8 mA, 50 pF) * Drive: 40 µA/.4 mA, 30pF 1 kΩ pullup
/BGACK	Out	Drive: 40 µA/.4 mA, 30 pF
SIZ0–SIZ1	(In) */Out	(Load: 40 µA/.4 mA, 30 pF) * Drive: 40 µA/.4 mA, 30 pF
/AS	(In) */Out	(Load: 300 µA/8 mA, 100 pF) * Drive: 40 µA/.4 A, 30 pF 1 kΩ pullup
/DSACK0–/DSACK1	In */Out *	Load: 100 µA/8 mA, 50 pF * Drive: 40 µA/.4 mA, 30 pF * 1 kΩ pullup *
R/W	(In) */Out	(Load: 300 µA/8 mA, 100 pF) * Drive: 40 µA/.4 A, 30 pF 1 kΩ pullup
/STERM	In/Out *	Load: 100 µA/8 mA, 50 pF Drive: 40 µA/.4 mA, 30 pF * 1 kΩ pullup
/CBACK	In */Out	Load: 100 µA/8 mA, 50 pF * Drive: 40 µA/.4 A, 30 pF 1 kΩ pullup
/CBREQ	In */Out	Load: 100 µA/8 mA, 50 pF * Drive: 40 µA/.4 mA, 30 pF 1 kΩ pullup
/CIOUT	(In)*/Out	Load: (100µA/8mA, 50 pF)* Drive: 40 µA/.4 A, 30 pF 1 kΩ pullup
/DS	(In) */Out	(Load: 100 µA/8 mA, 50 pF) * Drive: 40 µA/.4 mA, 30pF 1 kΩ pullup *
/RMC	(In) */Out *	(Load: 100 µA/8 mA, 50 pF) * Drive: 40 µA/.4 mA, 30pF 1 kΩ pullup *

(Continued)

■ **Table 18-4** Macintosh IIci cache connector signals, loading or driving limits (Continued)

Signal name	Input/output	Load or drive limits
/IPL0 –IPL2	Out *	Drive: 40 µA/.4 mA, 30pF * 1 kΩ pullup *
CPUCLK	Out	Drive: 10 µA/10 µA, 15 pF
/ROMOE	Out*	Drive: 40 µA/.4 mA, 30pF *
CPUDIS	In *	Load: 8 mA/1 mA, 30 pF * 1 kΩ pulldown *
CACHE	In	Drive: 8 mA/1 mA, 30 pF 1 kΩ pulldown
/CFLUSH	Out	Drive: 40 µA/.4 A, 30 pF
/CENABLE	Out	Drive: 40 µA/.4 A, 30 pF

* These signals are 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, .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 tri-state 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 kilohm resistor to pull /BGACK up to +5 volts, 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.

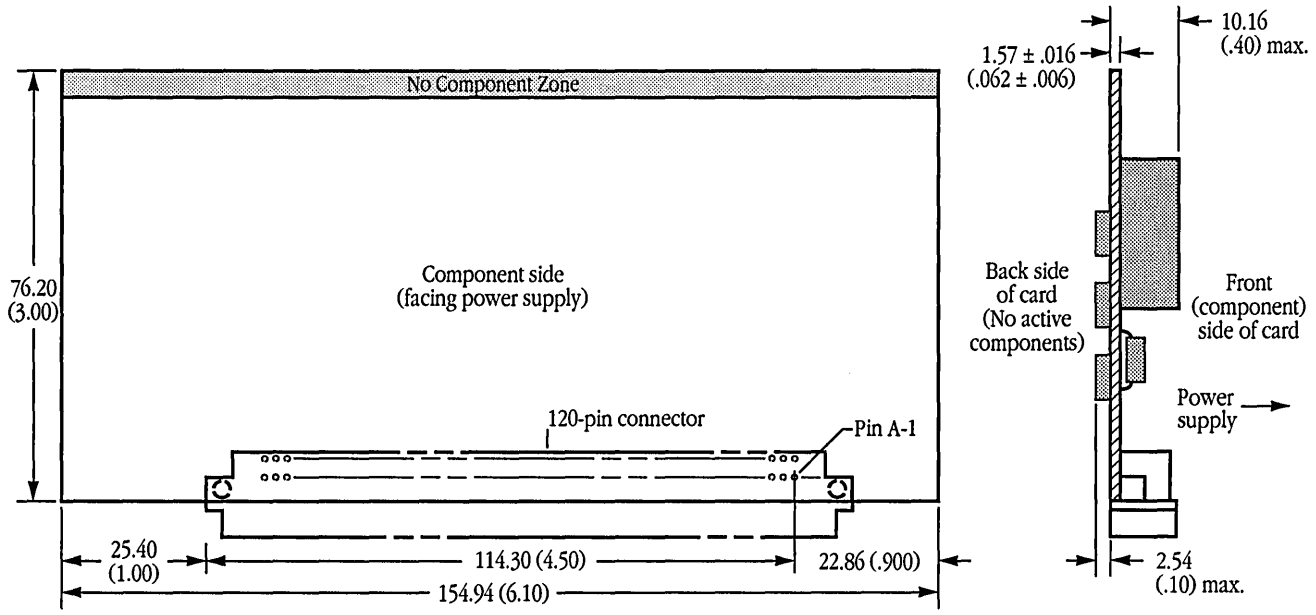
△ **Important** Although the cache expansion connector is capable of other functions, Apple plans to support its use for RAM cache cards only. △

Mechanical design guidelines for the cache card

Figure 18-2 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 0.062 ± 0.0075 inches.

■ **Figure 18-2** 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 in order to ensure proper thermal dissipation and eliminate the possibility of electrical interference with the ROM SIMM.

- Card warpage must be controlled to within a 0.10 inch deviation from the ideal.
- You should place no components or traces in the top 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 0.40 inches 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 0.10 inches 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 18-5 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 18-5** Power budget for a Macintosh IIci cache card

Device	+5 V	+12 V	–12 V
Cache card	1.0 amps	not available	not available
NuBus card	2.0 amps	0.175 amps	0.150 amps

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 watts versus the 7.5 watts specified for an expansion card in a conventional processor-direct slot.

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 processor-direct slot 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:

- an expansion card is mounted internally with no external I/O connections
- 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 pick-up 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 of 18 pF capacitance.
- 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 or Macintosh SE/30, 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 bandwidth 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 good quality connector, 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 processor-direct slot 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 the Macintosh II family of computers:

- Dissipation by a NuBus expansion card of up to 13.3 watts 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 watts 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 watts 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 air flow 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 processor-direct slot cards

You should follow these guidelines when designing an expansion card for a Macintosh computer with the processor-direct slot expansion interface:

- Dissipation by the expansion card of up to 7.5 watts of power provides a comfortable margin for the major computer components. Dissipation of more than 7.5 watts of power may cause excessive temperature rise on certain critical components. Apple studies indicate that at an ambient temperature of about 24°C, 7.5 watts 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 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 watts.
- 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 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 air flow 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 processor-direct slot 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 backwards 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 rather than 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 of 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 persons not knowledgeable of 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 **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 * A19D11L * /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
                    {holding terms}
            ;

/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
            vcc

.equations

    ARBCY       := MASTER * MASTERD * /OWNER * /ARBCY * /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 for 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 ADRCY 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 *  ARBCY * /DTACY
                           {we own bus, while not waiting for ACK}
                ;
    .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}
                ;
```

```
    .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 "offical" 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-ATTENTION 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 C PAL Listings 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  /RESET
        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
        /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

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 the Macintosh II family, 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: 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.

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.

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 II family.

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 bus is busy between start and acknowledge cycles.

byte lane: Any of four 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 II–family computer.

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.

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.

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 which 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.

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 that identifies the card and its functions, and 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 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 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 that appear to the user to be very fast, silent disk drives but 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.

format block: An element in a declaration ROM's firmware structure that provides a standard entry point for other elements in the structures. 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.

functional sResource: An sResource in an expansion card's declaration ROM that desribes 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; $9 through $B for the Macintosh IIcx; and $C through $E for the Macintosh IIci). 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 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 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, a longword consists 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.

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 the Macintosh II family of computers. See also **NuBus expansion slot** and **bus interface logic.**

NuBus expansion slot: A connector on the NuBus in a Macintosh II–family computer, into which an expansion card can be installed. The Macintosh II, Macintosh IIx, and Macintosh IIfx have six NuBus expansion slots; the Macintosh IIcx and the Macintosh IIci have three.

null cycle: A type of attention cycle that reinitiates bus arbitration.

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.**

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: 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.

pseudo-slot 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 II–family computers.

QuickDraw: The part of the Macintosh Toolbox that performs all graphic operations on the screen.

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 Systems 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 are electrically different. 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).**

slave: A card that responds to being addressed by another card acting as a master. The Macintosh II–family 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: A period of time during which 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 (the Macintosh II family provides from three 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 hex number corresponding to each card slot. For the Macintosh II family, 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 II–family ROM routines that communicate with an expansion card's declaration ROM and allow an application to gain access to declaration ROM.

slot space: The address space assigned to NuBus cards in Macintosh II–family computers and to PDS expansion cards that emulate NuBus cards in Macintosh SE/30 and Macintosh IIfx computers. See also **standard slot space, super slot space.**

sResource: 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) that 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 time 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: 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.

three-state: A bus driver that drives a line low or high or doesn't drive it at all.

timeout period: The time period that a bus master waits for a non-responding slave to respond before generating a bus timeout error code.

transaction: A complete NuBus operation such as read or write. In the Macintosh II family, 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 acknowledgement information.

transfer mode: One of the 16 modes or encodings that specify which part of the addressed 32-bit word is to be transferred.

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: 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 word is 16 bits long, a halfword, 8 bits. An MC68030 word is 32 bits long, a halfword, 16 bits.

Index

D

E

N

O

P

Q

R

S

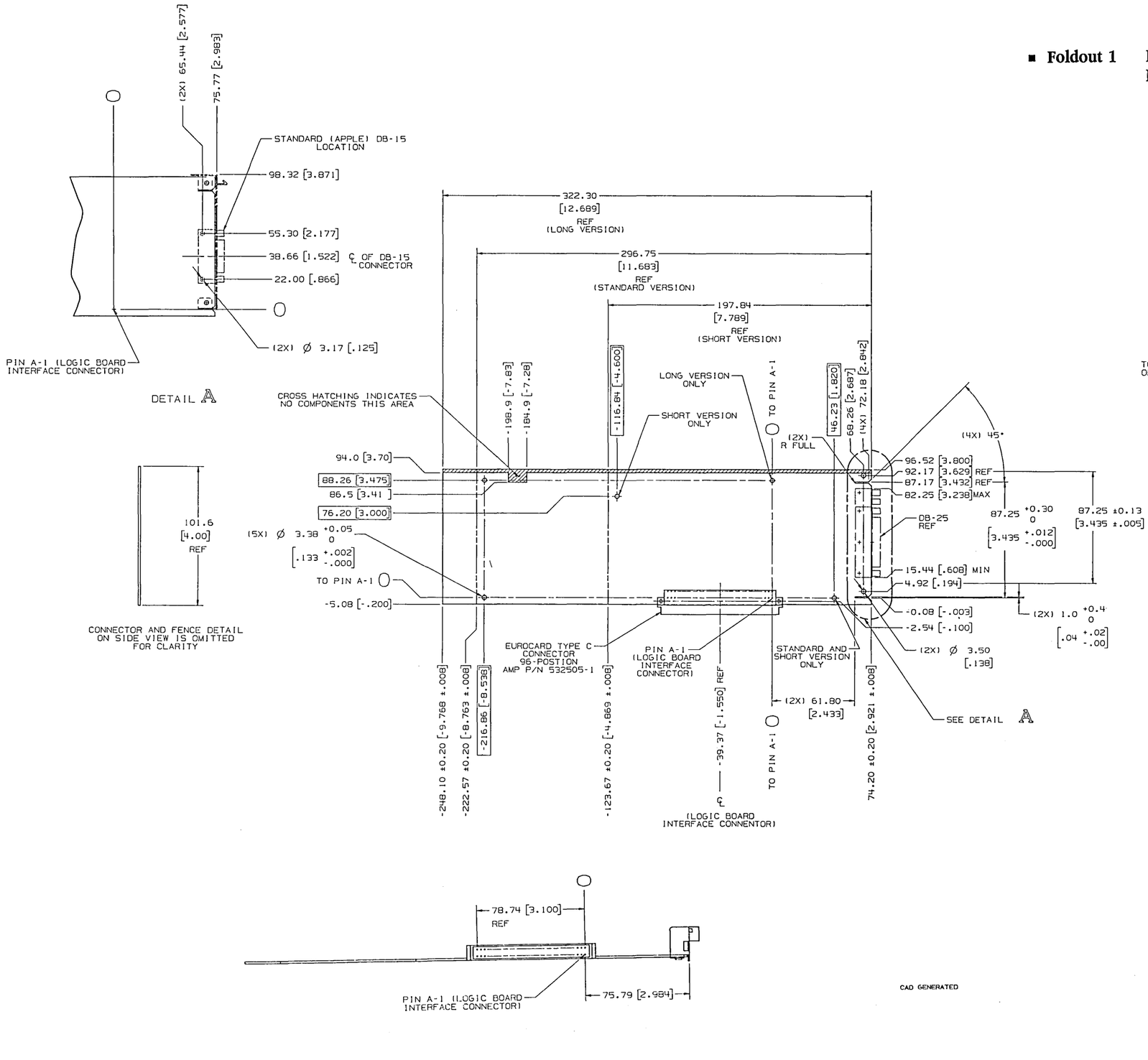
T

U

V

W

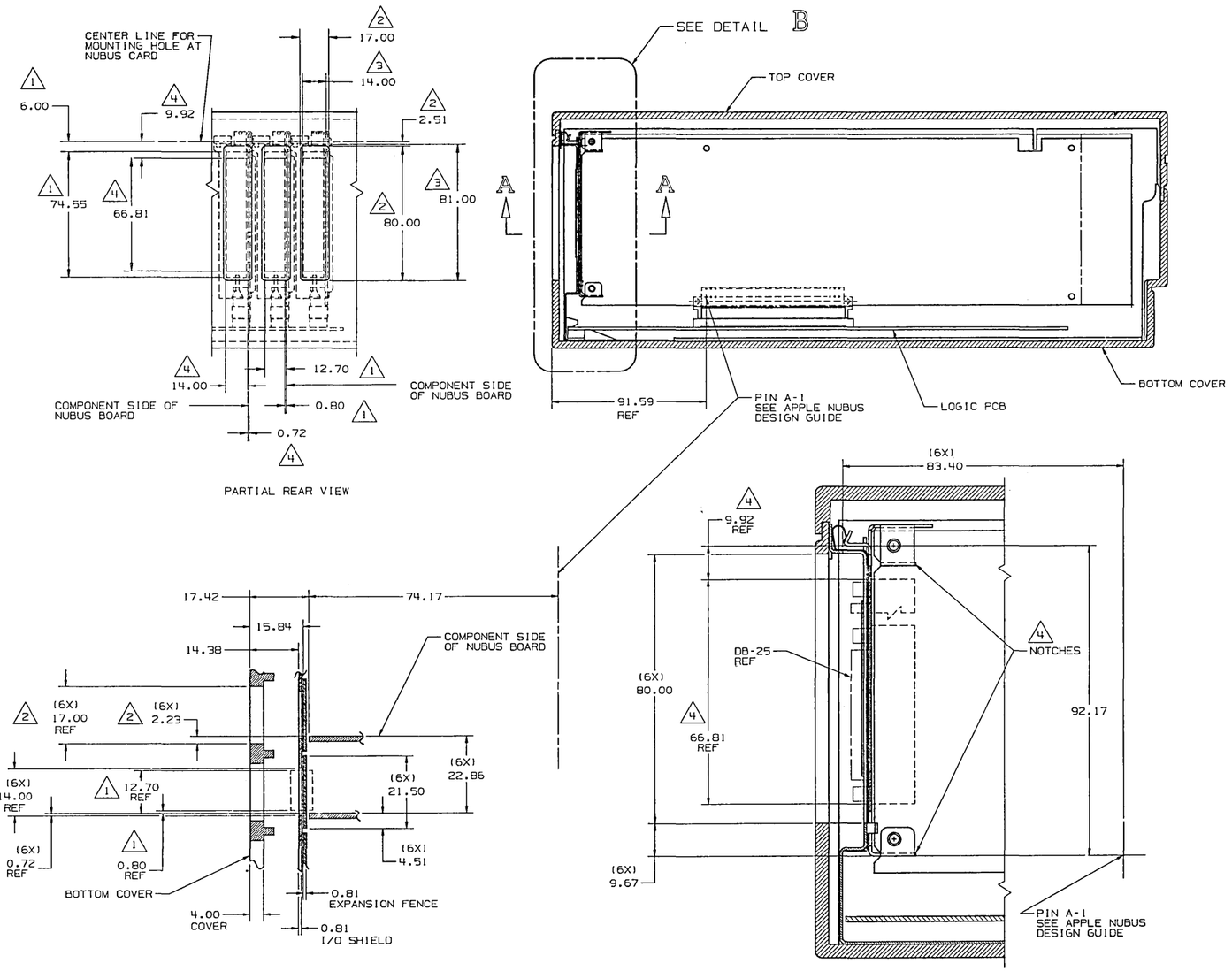
Foldouts



CAD GENERATED

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 INTO THE MAC II AND MAC IIx, 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 FOR THE MAC II AND MAC IICX. THESE SHOULD BE USED AS DESIGN LIMITS FOR MATING CABLES OR COMPONENTS THAT REQUIRE REAR ACCESS.
3. I/O SHIELD CLEARANCE DIMENSIONS FOR THE MAC II AND MAC IIX. 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 FOR MAC II AND MAC IIX. THESE SHOULD BE USED AS DESIGN LIMITS FOR APPLE SPECIFICATION 062-0484. NUBUS BOARDS WITH NOTCHES, REFERENCE DETAIL B.



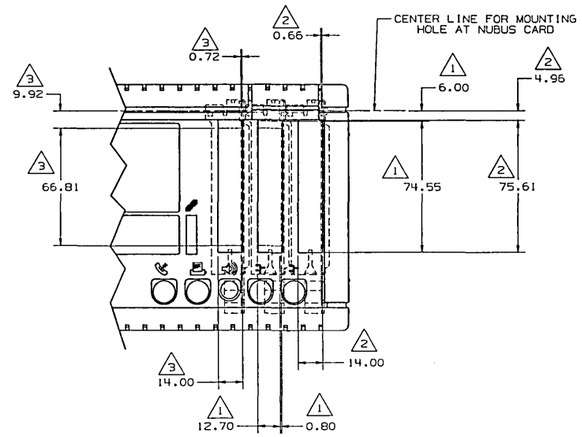
■ **Foldout 2** NuBus card clearance requirements, Macintosh II, Macintosh IIx, and Macintosh IIfx

NOTE: UNLESS OTHERWISE SPECIFIED

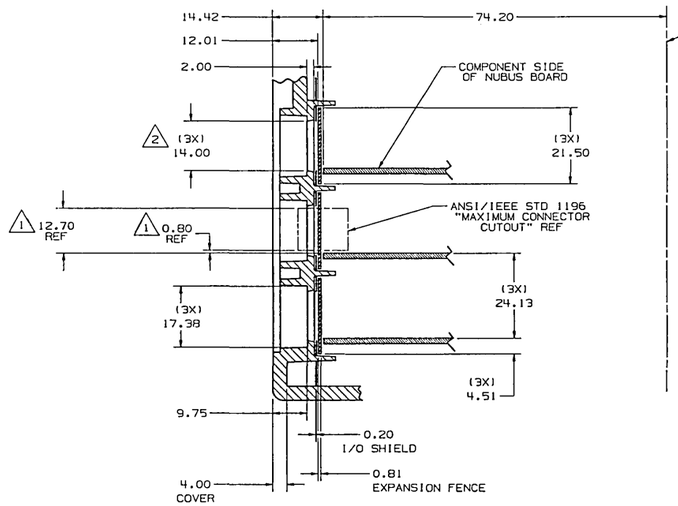
1. NUBUS BOARDS WHICH CONFORM TO THE ANSI/IEEE STD 1196 SPECIFICATION (AS SHOWN IN THE PARTIAL REAR), FOR "MAXIMUM CONNECTOR CUTOUT" WILL FIT PROPERLY INTO THE MAC IIcx, AS LONG AS THE DESIGN ALSO ALLOWS CLEARANCE FOR SURROUNDING PLASTIC. REFERENCE SECTION A-A AND DETAIL B.

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.

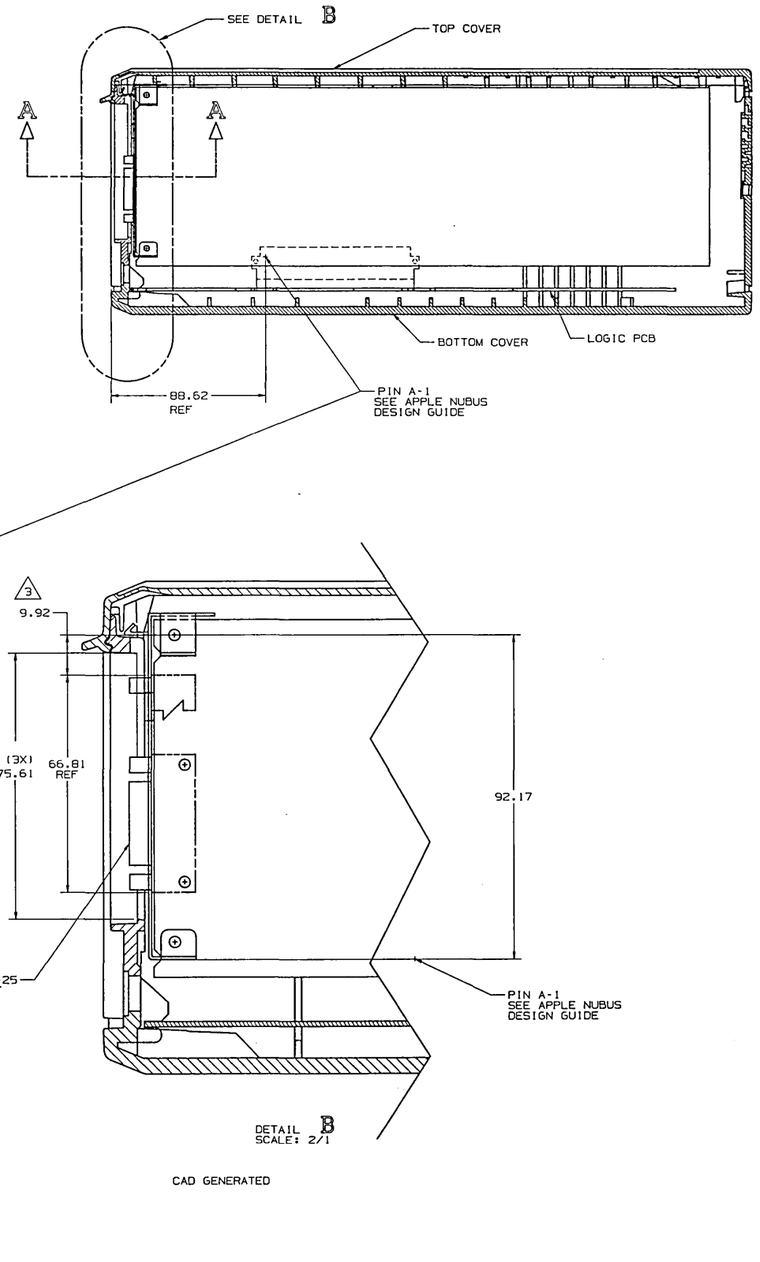
3. APPLE COMPUTER NUBUS BOARD CONNECTOR CLEARANCE DIMENSIONS FOR MAC IIcx. THESE SHOULD BE USED AS DESIGN LIMITS FOR APPLE SPECIFICATION 062-0484. NUBUS BOARDS WITH NOTCHES, REFERENCE DETAIL B.



PARTIAL REAR VIEW



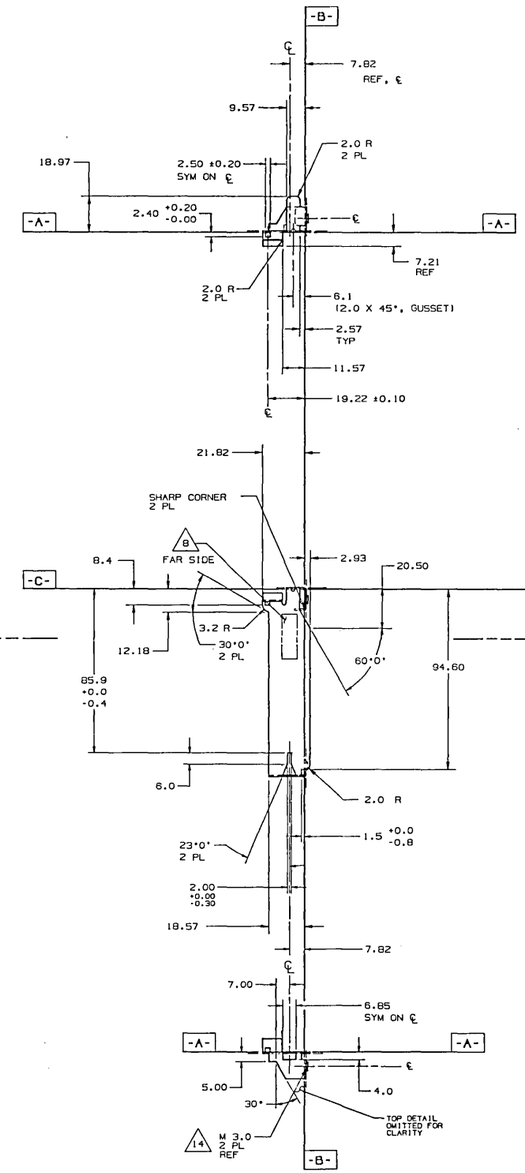
SECTION A-A
SCALE: 2/1

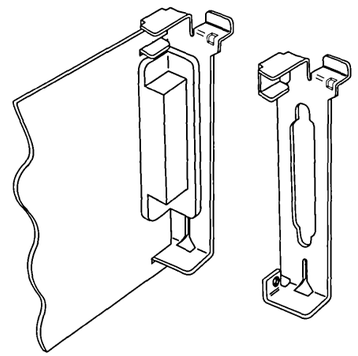


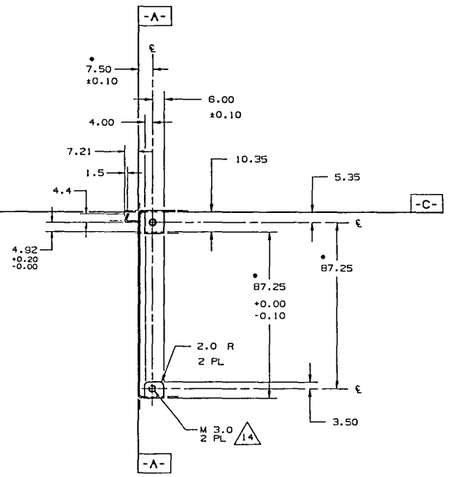
DETAIL B
SCALE: 2/1

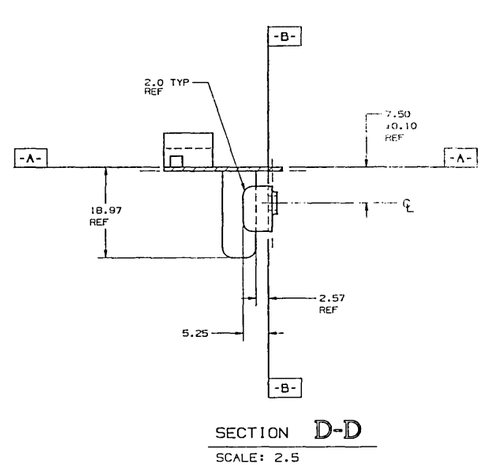
CAD GENERATED

■ **Foldout 3** NuBus card clearance requirements, Macintosh IIcx and Macintosh IIci

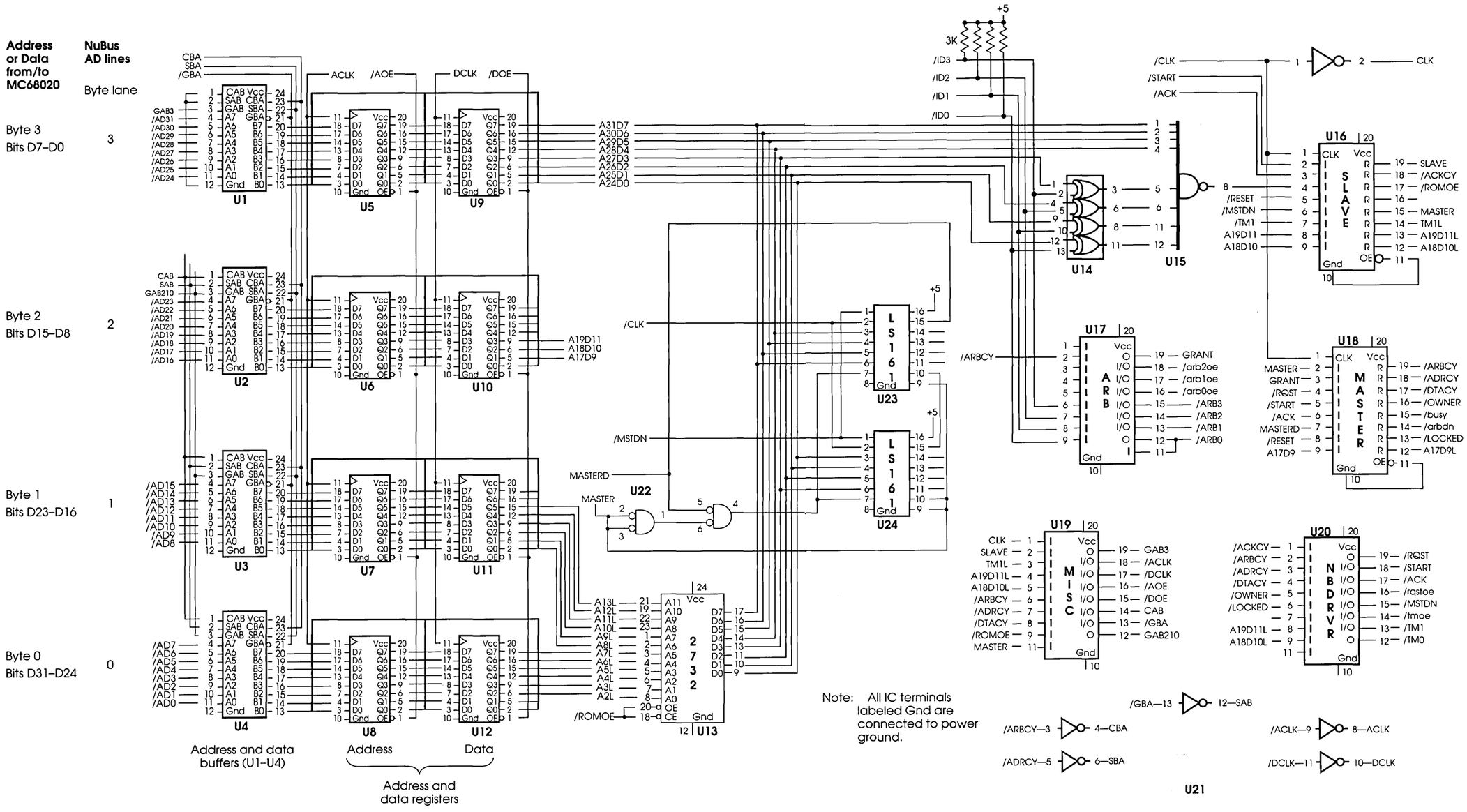


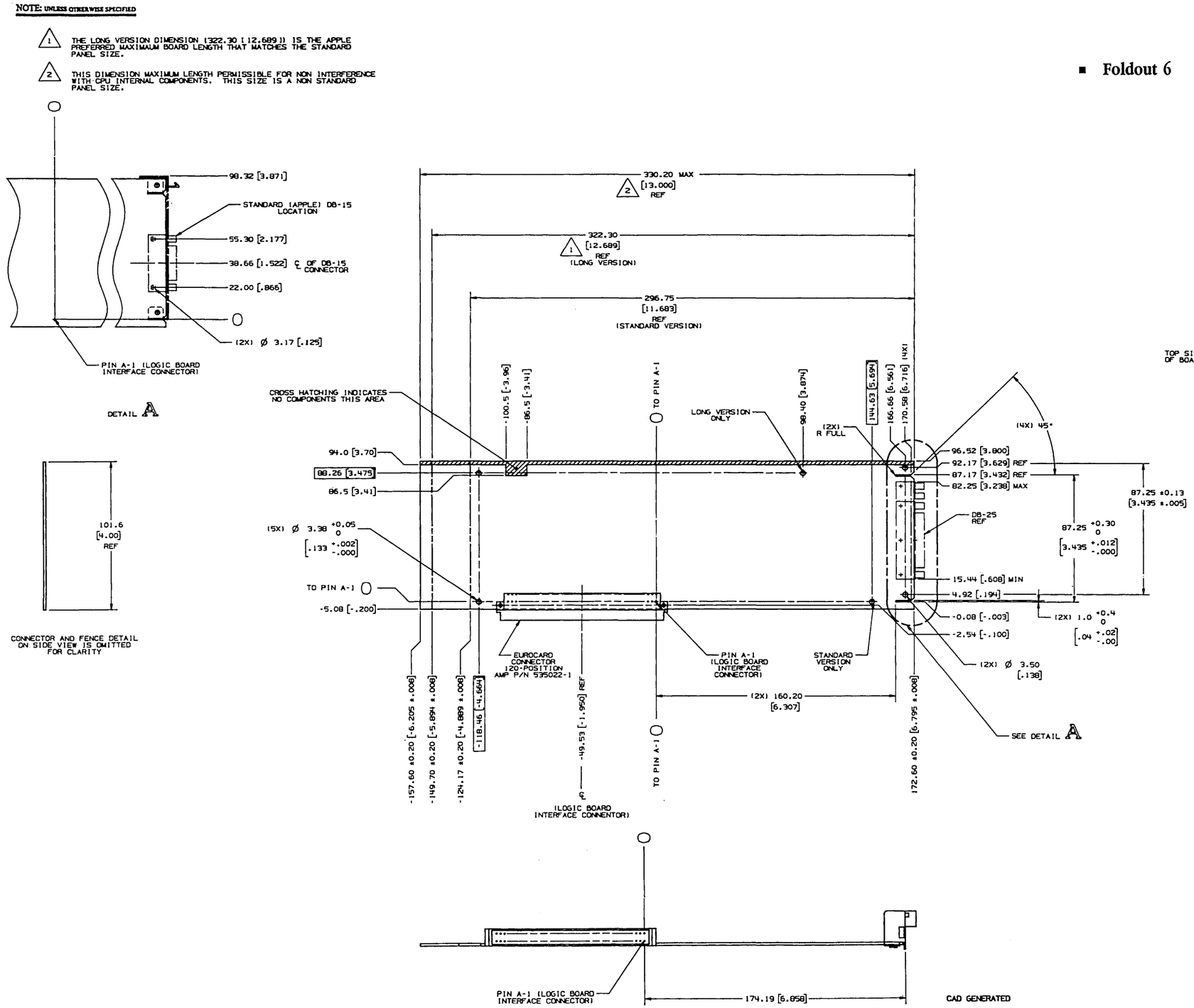






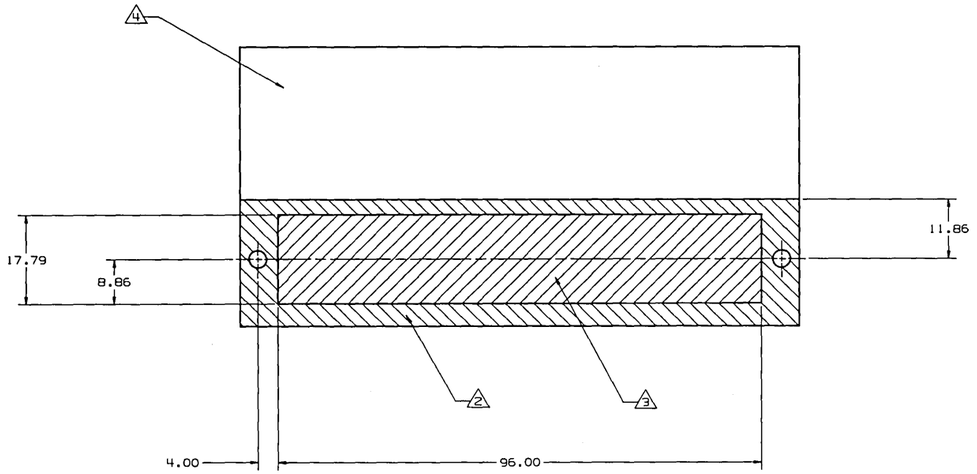
SECTION D-D
SCALE: 2.5


Address or Data from/to MC68020
NuBus AD lines
Byte lane
Byte 3
Bits D7–D0
3
Byte 2
Bits D15–D8
2
Byte 1
Bits D23–D16
1
Byte 0
Bits D31–D24
0
CBA
SBA
/GBA
ACLK
/AOE
DCLK
/DOE
U1
U2
U3
U4
U5
U6
U7
U8
U9
U10
U11
U12
A31D7
A30D6
A29D5
A28D4
A27D3
A26D2
A25D1
A24D0
A19D11
A18D10
A17D9
+5
3K
/ID3
/ID2
/ID1
/ID0
U14
U15
/CLK
/START
/ACK
CLK
U16
SLAVE
/RESET
/MSTDN
/TM1
A19D11
A18D10
SLAVE
/ACKCY
/ROMOE
MASTER
TM1L
A19D11L
A18D10L
/CLK
/MSTDN
MASTERD
MASTER
U22
U23
U24
U13
/ROMOE
U17
/ARBCY
GRANT
/arb2oe
/arb1oe
/arb0oe
/ARB3
/ARB2
/ARB1
/ARB0
U18
MASTER
GRANT
/RQST
/START
/ACK
MASTERD
/RESET
A17D9
/ARBCY
/ADRCY
/DTACY
/OWNER
/busy
/arbdn
/LOCKED
A17D9L
U19
CLK
SLAVE
TM1L
A19D11L
A18D10L
/ARBCY
/ADRCY
/DTACY
/ROMOE
MASTER
MISC
GAB3
/ACLK
/DCLK
/AOE
/DOE
CAB
GBA
U20
/ACKCY
/ARBCY
/ADRCY
/DTACY
/OWNER
/LOCKED
A19D11L
A18D10L
NBDRVR
/RQST
/START
/ACK
/rqstoe
/MSTDN
/tmoe
/TM1
/TM0
/GBA—13
12—SAB
/ARBCY—3
4—CBA
/ADRCY—5
6—SBA
/ACLK—9
8—ACLK
/DCLK—11
10—DCLK
U21
Address and data buffers (U1–U4)
Address
Data
Address and data registers
Note: All IC terminals labeled Gnd are connected to power ground.

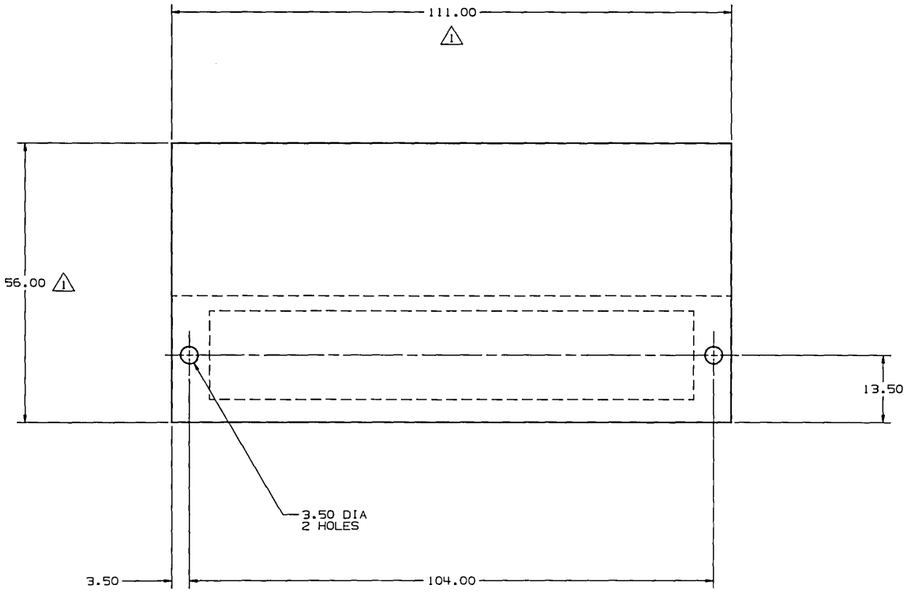

NOTE: UNLESS OTHERWISE SPECIFIED
1 THE LONG VERSION DIMENSION (322.30 [12.689]) IS THE APPLE PREFERRED MAXIMUM BOARD LENGTH THAT MATCHES THE STANDARD PANEL SIZE.
2 THIS DIMENSION MAXIMUM LENGTH PERMISSIBLE FOR NON INTERFERENCE WITH CPU INTERNAL COMPONENTS. THIS SIZE IS A NON STANDARD PANEL SIZE.
98.32 [3.871]
STANDARD (APPLE) DB-15 LOCATION
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
330.20 MAX
[13.000]
REF
322.30
[12.689]
REF
(LONG VERSION)
296.75
[11.683]
REF
(STANDARD VERSION)
CROSS HATCHING INDICATES NO COMPONENTS THIS AREA
100.5 [3.96]
86.5 [3.41]
TO PIN A-1
LONG VERSION ONLY
98.40 [3.874]
144.93 [5.707]
166.66 [5.561]
170.58 [6.716] (4X)
(2X) R FULL
(4X) 45°
94.0 [3.70]
88.26 [3.475]
86.5 [3.41]
96.52 [3.800]
92.17 [3.629] REF
87.17 [3.432] REF
82.25 [3.238] MAX
87.25 ±0.13
[3.435 ±.005]
(5X) Ø 3.38 +0.05 0 [.133 +.002 -.000]
DB-25 REF
87.25 +0.30 0 [3.435 +.012 -.000]
TO PIN A-1
15.44 [.608] MIN
4.92 [.194]
5.08 [.200]
-0.08 [.003]
(2X) 1.0 +0.4 0 [.04 +.02 -.00]
2.54 [.100]
(2X) Ø 3.50 [.138]
EUROCARD CONNECTOR 120-POSITION AMP P/N 535022-1
PIN A-1 (LOGIC BOARD INTERFACE CONNECTOR)
STANDARD VERSION ONLY
(2X) 160.20
[6.307]
SEE DETAIL A
157.60 ±0.20 [6.205 ±.008]
149.70 ±0.20 [5.894 ±.008]
124.17 ±0.20 [4.889 ±.008]
118.46 [4.664]
49.53 [1.950] REF
TO PIN A-1
172.60 ±0.20 [6.795 ±.008]
(LOGIC BOARD INTERFACE CONNENTOR)
PIN A-1 (LOGIC BOARD INTERFACE CONNECTOR)
174.19 [6.858]
CAD GENERATED
101.6
[4.00]
REF
CONNECTOR AND FENCE DETAIL ON SIDE VIEW IS OMITTED FOR CLARITY
22.86 [.900]
TYP
TOP SIDE OF BOARD
CPU LOGIC BOARD/PERIPHERAL BOARD SPACING (3 BOARDS SHOWN FOR CLARITY)

NOTE: UNLESS OTHERWISE SPECIFIED

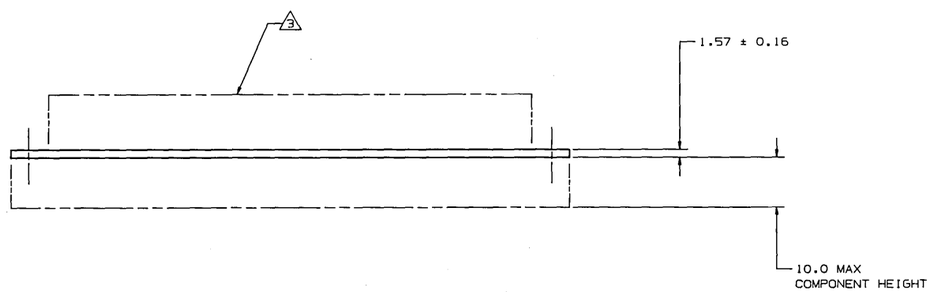
1. INDICATED DIMENSIONS SPECIFY MAXIMUM BOARD OUTLINE. NO COMPONENTS, INCLUDING MATING CONNECTOR FROM EXPANSION BOARD, TO PROTRUDE BEYOND THE BOARD EDGES.
2. INDICATED AREA WILL CONTACT THE METAL CHASSIS WHEN BOARD IS INSTALLED. NO TRACES OR FEEDTHROUGHS ARE PERMITTED.
3. INDICATED AREA IS RESERVED FOR 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



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 Apple LaserWriter® printers. Final pages were created on the Varityper VT600W® printer. Line art was created using Adobe Illustrator™. PostScript®, the page-description language for the Laser Writer, was developed by Adobe Systems Incorporated.

Text type and display type are Apple's corporate font, a condensed version of Garamond. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

Writer: Rolly Reed
Developmental Editor: Laurel Rezeau
Art Direction: Deb Dennis
Illustration: Mil Madamba
Production: J. Renee Ekleberry, Janet M. Anders

Special thanks to Mark Baumwell, Rich Collyer, Jon Fitch, David Fung, Denis Hescox, Ron Hochsprung, Brian Howard, Jon Krakower, Ann Nunziata, Noah Price, and Jim Stockdale for their technical assistance.

Also, a very special thanks to Roy Smith, writer of the first edition, *Designing Cards and Drivers for Macintosh II and Macintosh SE.*

The Official Publication from Apple Computer, Inc.

Designing Cards and Drivers for the Macintosh® Family
Second Edition

Written and produced by the people at Apple Computer, Inc., this is the official guide for developers of expansion cards and peripheral devices for the open-architecture Apple® Macintosh computers. A companion book, *Guide to the Macintosh Family Hardware*, second edition, provides detailed information on the hardware design of the Macintosh family of computers.

Designing Cards and Drivers for the Macintosh Family, second edition, provides comprehensive expansion card design and driver design guidelines for eight Macintosh models: Macintosh SE, Macintosh SE/30, Macintosh II, Macintosh IIx, Macintosh Portable, Macintosh IIcx, Macintosh IIci, and Macintosh IIfx.

The book consists of an introduction, describing Apple's expansion strategy for current and future Macintosh products, and three main parts.

Part I describes the implementation of the NuBus™ interface in the Macintosh II family of computers. It provides hardware designers with electrical and mechanical guidelines for designing NuBus expansion cards, and supplies programmers with information that is essential to the design of declaration ROM and driver software.

Part II describes the processor-direct slot (PDS) expansion interface and explains its relationship to the microprocessor. It provides hardware designers with electrical and mechanical guidelines for designing expansion cards for the 68000 Direct Slot and 68030 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.

An appendix provides guidelines for reducing electromagnetic interference and for complying with heat dissipation and product safety standards in your expansion card designs. The book also contains a glossary of technical terms and an index.

Printed in U.S.A.

Apple Computer, Inc.
20525 Mariani Avenue
Cupertino, California 95014
408-996-1010
TLX 171-576

Addison-Wesley Publishing Company, Inc.

ISBN 0-201-52404-X
52404