

Cortland Hardware Reference

Beta Draft
Engineering p/n: 030-1290-PN8
Finance p/n: pAP002-16
September 19, 1986

Writer: Dave Carpenter
User Education,
Technical Publications

Cortland Hardware Reference Manual

🍏 APPLE COMPUTER, INC.

This manual is copyrighted by Apple or by Apple's suppliers, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple Computer, Inc. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

© Apple Computer, Inc., 1986
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

Apple and the Apple logo are registered trademarks of Apple Computer, Inc.

Simultaneously published in the United States and Canada.

Changes Since Previous Draft

This is the alpha draft of the Cortland Hardware Reference manual, and is based primarily on the following ERS's:

- Mega // ERS 2.2, June 25, 1985
- Fast Processor Interface (FPI) ERS, March 25, 1986
- Video Graphics Controller (VGC) ERS 2.1, November 19, 1985
- Slot Maker ERS, May 1, 1985
- Keyglu ERS 1.0, June 25, 1985
- Apple DeskTop Bus ERS Rev B, June 13, 1985
- Ensoniq ERS Rev 1.0, June 25, 1986
- Sound Glu ERS, June 21, 1985
- Cortland Sound ERS Rev 1, June 25, 1986

Several changes have occurred since the Alpha draft. The most significant change is in the composition of chapter 2. In the alpha draft, this chapter contained information about the Mega II proper, but has now been greatly expanded to include much information lifted from the Apple IIe technical reference manual. The chapter is now titled "Chapter 2: The Mega II: Maintaining Compatibility". Other significant changes include:

- Sound GLU and Sound DOC chapters are now combined into one titled "Chapter 6: Cortland Sound".
- Keyboard GLU and Front Desk Bus chapters are now combined into one titled "Chapter 7: Apple DeskTop Bus".
- The FPI chapter is now titled "Chapter 3: The FPI: New Hardware Features".
- New chapters were created and added to the Beta draft:
 - A chapter about the expansion slots was added and titled: "Chapter 5: Peripheral Expansion".
 - A chapter about the disk drive port was added and titled: "Chapter 8: The Disk Port".
 - A chapter was added about the 65C816 microprocessor was added.
- A glossary accompanies this draft.

Cortland Hardware Reference Manual

In general, information in each chapter was examined and determined whether or not it was appropriate to be located in that chapter. This resulted in much interchanging of information between chapters.

Finally, the Beta draft period consisted infinite instances of rewriting sentences ("active voice, active voice!": P. Brown), rearranging of order of information ("o.k. to put this *before* that?": L Aochi), deleting of impertinent information ("why do we need to know *this*?": D. Bice) and addition of incomplete information ("*what*?": R. Carr).

Thanks to those above and those nameless, countless others who so kindly reviewed the alpha draft and helped to create this, the new, the improved, the **BETA draft!**

Contents

6	Foreword
7	Chapter 1: Introduction To The Cortland
7	Introduction
9	The "Old" Cortland
9	The "New" Cortland
10	Operation
10	Memory Allocation
11	Summary
	Chapter 2: Mega II: Maintaining Compatibility
19	Introduction
20	The Mega II Custom IC
21	The Apple IIe
21	RAM Upgrade
22	Apple I/O
22	The keyboard
23	Reading the keyboard
24	Video
24	The video display generator
27	Text modes
27	Text character sets
28	40-column versus 80-column text
30	Graphics modes
30	Low-resolution graphics
31	Hi-resolution graphics
33	Double high-resolution graphics
35	Video display pages
36	Display mode switching
38	Addressing display pages directly
45	The text window
46	Secondary inputs and outputs
46	The speaker
47	Game I/O
47	The hand control connector signals
48	Annunciator outputs
48	Strobe output
49	Switch inputs
49	Analog inputs
49	Summary of secondary I/O locations
50	Memory
50	Memory organization
51	Main memory map
52	RAM memory allocation
53	Reserved memory pages
53	Page zero
53	The 65C02

Cortland Hardware Reference Manual

54	The input buffer
54	Link-address storage
54	The display buffers
55	Bank-switched memory
56	Setting bank switches
58	Reading bank switches
59	Auxiliary memory
61	Memory mode switching
64	Peripheral Expansion
65	Peripheral-card memory spaces
65	Peripheral-card I/O space
66	Peripheral-card ROM space
67	Expansion ROM space
68	Peripheral-card RAM space
69	I/O programming suggestions
70	Finding the slot number with ROM switched in
70	I/O addressing
72	RAM addressing
72	Other uses of I/O memory space
73	Switching I/O memory
74	Developing cards for slot 3
75	Pascal 1.1 firmware protocol
75	Device identification
76	Routine entry points
77	Interrupts
78	What is an interrupt?

Chapter 3: The FPI: New Hardware Features

79	Introduction
80	The FPI Subsystem
80	Standard Soft Switches
81	Memory Allocation
81	The State Register
82	Shadowing
83	The Shadow Register
85	The Configuration Register
86	RAM Control
87	ROM
87	I/O Processing
87	The Slot register
89	Synchronization
90	The Mega II Cycle
90	Mega II Aux Bank Access
91	Real Time Clock IC Interface

Chapter 4: Video

93	Introduction
93	Apple II Compatibility
94	The Video Graphics Controller
95	Video Display Enhancements

95	Text and Background Color
96	Border Color
97	To Color or Not to Color...
97	New Graphics Display Modes
98	Super Hi-Res Graphics
98	Super Hi-Res Graphics Buffer
99	Pointer Bytes
100	Color Palettes
101	Pixels
103	Color-Fill Mode
104	VGC Interrupts
104	VGC Interrupt register
106	VGC Interrupt-Clear register
106	Graphics Summary

Chapter 5: The Expansion Slots

109	Introduction
110	The Expansion slots
113	Apple II Compatibility
113	Direct Memory Access
113	Interfacing the Cortland System
114	Slot I/O Cards
114	DMA Cards
115	The Buffered Address Bus
117	The Slot Data Bus
119	Interrupt and DMA Daisy Chains
122	Loading and Driving Rules
122	Summary

Chapter 6: Cortland Sound

123	Introduction
123	Sound Synthesis
124	Accessing the DOC
125	The Sound Control register
126	Address Pointer register
126	Write operation
126	Read operation
126	Playing back the sound
127	DOC Registers
127	The Oscillator Interrupt register
128	The Oscillator Enable register
128	The A/D Converter register
128	The Oscillator Control register
130	The Data register
130	The Volume register
130	The Frequency High and Frequency Low registers
130	The Waveform register
133	Summary
135	Ensoniq DOC Data Sheet

Chapter 7: Apple DeskTop Bus

- 137 Introduction
- 138 The input bus
- 138 Keyboard Microcontroller
- 139 The Keyboard GLU
 - 139 Keyboard GLU registers
 - 140 Keyboard/Mouse Status register
 - 140 Keyboard Data register
 - 142 Keyboard Microcontroller registers
 - 142 Modifier Key register
 - 143 Mouse Data register
 - 143 Command Status register
- 144 Bus Communication
 - 145 Signals
 - 145 Attention and Sync
 - 145 Reset
 - 146 Service Request
 - 146 Reset
 - 146 Transactions
- 146 Apple Desktop Bus Peripherals
- 146 Commands
 - 148 Talk
 - 148 Listen
- 148 Device registers
- 148 Collision Detection
- 148 Error Conditions
- 149 Network Layer (ADB Types)
 - 149 Normal Devices
 - 149 Extended Address Devices
 - 150 Register 3
 - 150 Service Request

Chapter 8: The Disk Port

- 153 Introduction
- 153 Apple II Compatibility
- 154 The Disk Port Connector
- 154 The IWM
- 155 The Disk Interface register

Chapter 9: The Memory Expansion Slot

- 157 Introduction
- 157 Extended RAM
- 158 Extended RAM Mapping
- 159 MSIZE
- 160 Ghosting
- 160 Extended ROM
- 160 Address Multiplexing

Chapter 10: Power Supply

- 161 Introduction
- 161 Function
- 161 Specifications
- 162 Power Connector

65C816 Microprocessor

- 163 Introduction
- 164 65C816 Features
- 165 The sixteen-bit 65C816
- 166 Microprocessor differences
- 166 The registers
 - 167 The Accumulator
 - 167 X Index register
 - 167 Y Index register
 - 167 Data Bank register
 - 167 Stack Pointer register
 - 167 Program Status register
 - 167 Program Counter register
 - 167 Program Bank register
- 168 Direct register
- 168 Emulating the 65C02
 - 168 The e bit
 - 168 The m bit
 - 168 The x bit
- 169 Operating speed
- 169 Summary
- 170 65C816 Data Sheet

A-1 Appendix A: Roadmap to the Cortland Technical Manuals

B-1 Appendix B: International Keyboards

C-1 Appendix C: Character Generator

D-1 Appendix D: Schematic Diagrams

E-1 Appendix E: Conversion Tables

107 Appendix F: Frequently Used Tables

Glossary

Foreword

About this manual

This is the hardware reference manual for the Cortland computer. It is for hardware designers and programmers who want to know how to manipulate the hardware, but will be useful to anyone wanting to know how to take advantage of all of its features.

As part of the Cortland Technical Suite of manuals, the *Cortland Hardware Reference* manual covers the design and function of the Apple Cortland hardware function. It provides hardware design and interface information and is intended for those individuals who have an understanding of digital microprocessor electronics and who are interested in interfacing the Cortland to the outside world. Those programmers who are using the sound and graphics capabilities of the Cortland should also refer to this manual so that they can have a good understanding of how software affects the hardware.

While the Cortland is a new computer, it is above all, an Apple II. This manual introduces the new, more powerful computing features, and also describes how it maintains compatibility with previous Apple II models.

Specifically, this manual

- introduces the new features of the Cortland hardware.
- describes the Apple II-compatible functions.
- provides addresses where necessary for manipulating the hardware
- provides input/output information for interfacing signals to the outside world.

Unlike previous Apple technical reference manuals, this one does not document firmware features; they are covered in detail in *The Cortland Firmware Reference* manual. The Cortland uses a toolbox of low-level utilities to perform certain functions. Programmers are expected to use these tools to perform sound and graphics operations whenever available.

How to use this manual

Each chapter in this manual describes a different aspect of the computer. Each chapter is modular in scope, describing the particular aspect of the Cortland, making references to other chapters and manuals where necessary.

Chapter 1: introduces the Cortland as a new Apple II computer.

Chapter 2: *Maintaining Compatibility* continues by describing how the Cortland acts like a standard Apple IIe.

In contrast, Chapter 3 introduces the new capabilities of the Cortland.

Chapter 4: *Video* goes in depth into how to make the new Super Hi-Res graphics work for you.

Chapter 5: *Peripheral Expansion* provides descriptions of the I/O slots and the signals available at an expansion slot. DMA and interrupts are described here also.

Chapter 6: *Cortland Sound* shows you how to control the 32 digital oscillators and generate sound.

Chapter 7: *Apple DeskTop Bus* provides details of the hardware and protocol required to design and connect an input device (keyboard, mouse, graphics tablet, etc.) to this input device.

The built-in disk drive port is described in Chapter 8.

Chapter 9 goes into detailed description of the memory expansion slot and how to design and access a memory expansion card for this special slot.

Chapter 10 briefly describes the Cortland power supply and lists its specifications.

The new 65C816 microprocessor is covered in Chapter 11.

Appendix A contains a Roadmap to the Cortland technical suite manuals. Read this appendix to determine which books you need to reference to learn more about a programming language, the Cortland firmware, or other aspect of the computer.

Appendix B has all nine international keyboard layouts.

Appendix C shows you the contents of the character generator, all the characters the Cortland can display.

Appendix D contains the schematic diagrams showing all of the electrical components of the main circuit board.

Appendix E has conversion tables; what a bit and a byte can represent. Conversion tables between hexadecimal, decimal and negative decimal as well as eight-bit ASCII are provided.

Appendix F contains some of the most frequently used tables taken from throughout the manual.

A glossary follows the appendixes.

Some Terminology

The Apple II Apple II plus are *standard* Apple II's. In this manual, reference is made of the Cortland's compatibility with standard Apple II's. This means that the Cortland will emulate an Apple II or Apple II plus computer. A particular function of the Cortland that is in common with the Apple IIe or Apple IIc, for instance, will be mentioned specifically as such. Also, software that runs on a standard Apple II computer is *standard* Apple II software.

Chapter 1

Introduction to the Cortland

The Cortland is a new computer in the Apple II family. While maintaining it's roots in the Apple //e and Apple //c, this new processor also provides new features which make it the most powerful Apple II yet. This first chapter describes generally how the new Cortland fits into the Apple II family of computers, and what sets it apart from previous Apple II products.

PHOTOGRAPH OF THE CORTLAND

Photo 1-1. The Cortland.

Removing the cover

The Cortland uses a two-piece case. The cover is hinged at the front, and is secured at the rear where the upper and lower halves meet. A snap-lock is located at either side of the rear panel as shown in Photo 1-2. To remove the cover, press in on each snap-lock while lifting up at the rear of the cover. Pivot the cover at the front, and remove it completely. The main circuit board is now exposed for access to the expansion slots. Photo 1-3 shows the major components of the Cortland.

PHOTO OF SNAP-LOCKS AT THE REAR OF THE COVER.

Photo 1-2. Releasing the snap-locks to remove the cover.

AERIAL PHOTO OF CORTLAND, COVER REMOVED. CALLOUTS.

Photo 1-3. The Cortland with cover removed.

Expansion slots

The Cortland, like the Apple //e, has seven peripheral expansion slots at the rear of the circuit board. These will accept most Apple II-compatible peripheral expansion cards designed for any of the Apple II processors. Note that the Cortland does not have an auxiliary slot as is found in the Apple //e. For more information on the peripheral expansion slots, see *Chapter 5: Peripheral Expansion*.

Connectors

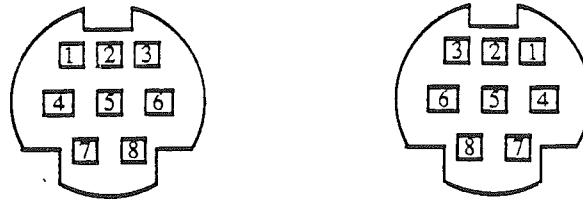
At the rear of the Cortland are several connectors. These connectors allow the computer to be connected an input device such as a keyboard or a mouse, or a peripheral device such as a disk drive, a printer, a modem, a network or the like. Photo 1-4 shows the connectors.

PHOTO OF REAR PANEL OF CORTLAND WITH CALLOUTS

Photo 1-4. The Cortland connectors.

Two serial ports: The two RS-232-C-compatible serial ports use mini-DIN 8-pin connectors. Power is supplied (+5 and ground) at these connectors, but the device connected to each serial port must not draw more than x milliamps. To transmit and receive data to and from a device connected to a serial port, use the firmware calls in the system ROM. Figure 1-1 shows the pin configuration of the serial ports.

To read about how to use the firmware in the Cortland ROM, refer to the *Cortland Firmware Reference manual*.



Serial port host connector Serial peripheral connector

Figure 1-1. Pin configuration of a serial port connector.

Pin	Description
1	DTR Data Terminal Ready
2	HSKI
3	TX Data Transmit data
4	Ground Ground reference and supply
5	Ground Ground reference and supply
6	RTS Ready To Send
7	GPI
8	RX Data Receive data

Disk drive port: This connector will accept either 2.5-inch or 3.5-inch Apple disk drives made for the Apple II. This connector is similar in function to the one on the Apple //c. The connector is a type DB-19. For more information on the disk drive port, see *Chapter 9: The Disk Port*. Figure 1-2 shows the pin configuration of the disk drive port.

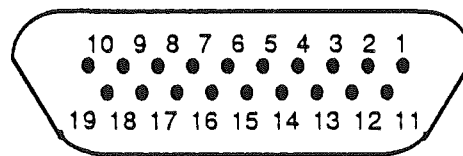


Figure 1-2. The disk port connector.

Pin	Description	Description
1,2,3	GND	Ground reference and supply
4	3.5DISK	3.5- or 5.25-inch drive select
5	-12V	-12 volt supply
6	+5V	+5 volt supply
7,8	+12V	+12 volt supply
9	DR2	Drive 2 select
10	WRPROTECT	Write protect input
11	PH0	Motor phase 0 output
12	PH1	Motor phase 1 output
13	PH2	Motor phase 2 output
14	PH3	Motor phase 3 output
15	WREQ	Write request
16	HDSEL	Head select
17	DR1	Drive 1 select
18	RDDATA	Read data input
19	WDATA	Write data output

RGB video connector: This connector provides analog red, green and blue (RGB) video signals for an analog-input-type RGB monitor. Do not use any type of RGB monitor other than an analog input-type. The connector is a type DB-25. See *Chapter 4: Video* for more information. Figure 1-3 shows the connector and the pin configuration of the RGB video connector.

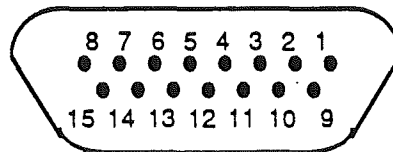


Figure 1-3. The RGB video connector.

Pin	Description	
1	Ground	Ground reference and supply
2	Red	Red analog video signal
3	Comp	Composite synch signal
4	N.C.	No connection
5	Green	Green analog video signal
6	Ground	Ground reference and supply
7	N.C.	No connection
8	+12V	+12 volt supply
9	Blue	Blue analog video signal
10	N.C.	No connection
11	Sound	Analog sound output
12	NTSC/PAL	Composite video output
13	Ground	Ground reference and supply
14	N.C.	No connection
15	N.C.	No connection

Composite video connector: Composite color video is available at this connector. A standard Apple composite color monitor can be used to display video. A television may be used to display 40-column text or graphics: this requires a video modulator to connect the Cortland to a television.

Apple DeskTop Bus: Connect Apple DeskTop Bus (ADB) devices to this connector. These devices may be ADB keyboards, ADB mice, ADB graphics tablets, or any other input device designed to the ADB specification. Do not attempt to adapt input devices not designed for ADB to this connector. Only keyboards, mice and other input devices designed specifically for ADB may be connected here. See *Chapter 8: Apple DeskTop Bus* for more information. Figure 1-4 shows the pin configuration of the ADB connector.

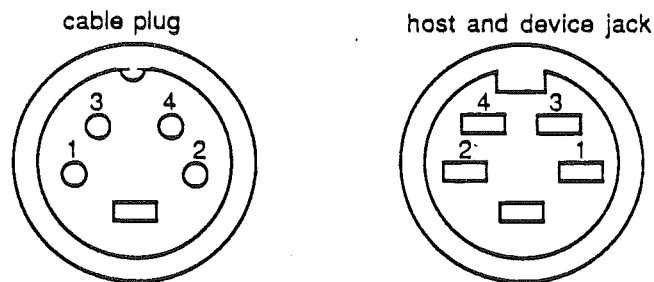


Figure 1-4. The ADB connector.

Pin	Description
1	data
2	reserved
3	+5 power supply @ 500 mA for all devices
4	signal and power ground

Mouse connector: Connect a standard Apple II mouse to this connector. Do not adapt an ADB mouse to connect to this connector. These two mice are completely different, and should not be interchanged. Figure 1-5 shows the pin configuration of the mouse connector.

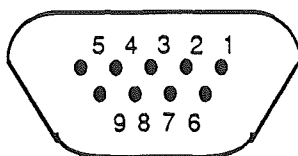


Figure 1-5. The mouse connector.

1	Ground
2	X-axis signal
3	+5 volts
4	Y-axis
5	N.C.
6	N.C.
7	N.C.
8	N.C.
9	N.C.

A closer look

You can think of the Cortland system as containing two separate and unique subsystems. These subsystems are not mutually exclusive; to the contrary, the subsystems share several components without which they could not function. In particular, both share the microprocessor, I/O, memory, video, and peripheral expansion circuitry.

The first subsystem consists of the parts of the computer that make the Cortland compatible with other Apple II products. These are

- the 65816 microprocessor
- the Mega II custom integrated circuit (IC)
- 128K of standard Apple II memory
- the Video Generator Chip (VGC) and video generation circuitry
- built-in peripherals and external input/output (I/O) slots

This subsystem is referred to as the Mega II portion of the system, after the controlling device. Figure 1-1, a block diagram of the Cortland, shows the Mega II subsystems and its relationship to the rest of the system.

The second subsystem consists of components of the computer that are new to the Apple II family. These are

- the 65816 microprocessor
- the Fast Processor Interface (FPI) custom integrated circuit (IC)
- 128K (expandible to 8M bytes) of dynamic RAM
- 128K (expandible to 1Megabyte) of ROM
- Digital Oscillator Chip (DOC) sound synthesizer and support circuitry

This subsystem is referred to as the FPI portion of the system, because of the controlling device, the FPI.

Note that the 65C816 microprocessor is listed as a component of both subsystems. Being a new microprocessor, it has many new instructions. This makes it a powerful new device which allows new capabilities of this new computer. Also, the 65C816 recognizes the complete 65C02 instruction set, which means it will run most existing Apple II software.

Figure 1-6 shows the Cortland computer in block form. Note the dotted division separating the two subsystems. While this is a logical division, it is not absolute: the FPI portion has access to the peripheral slots, the VGC and other components on the Mega II side.

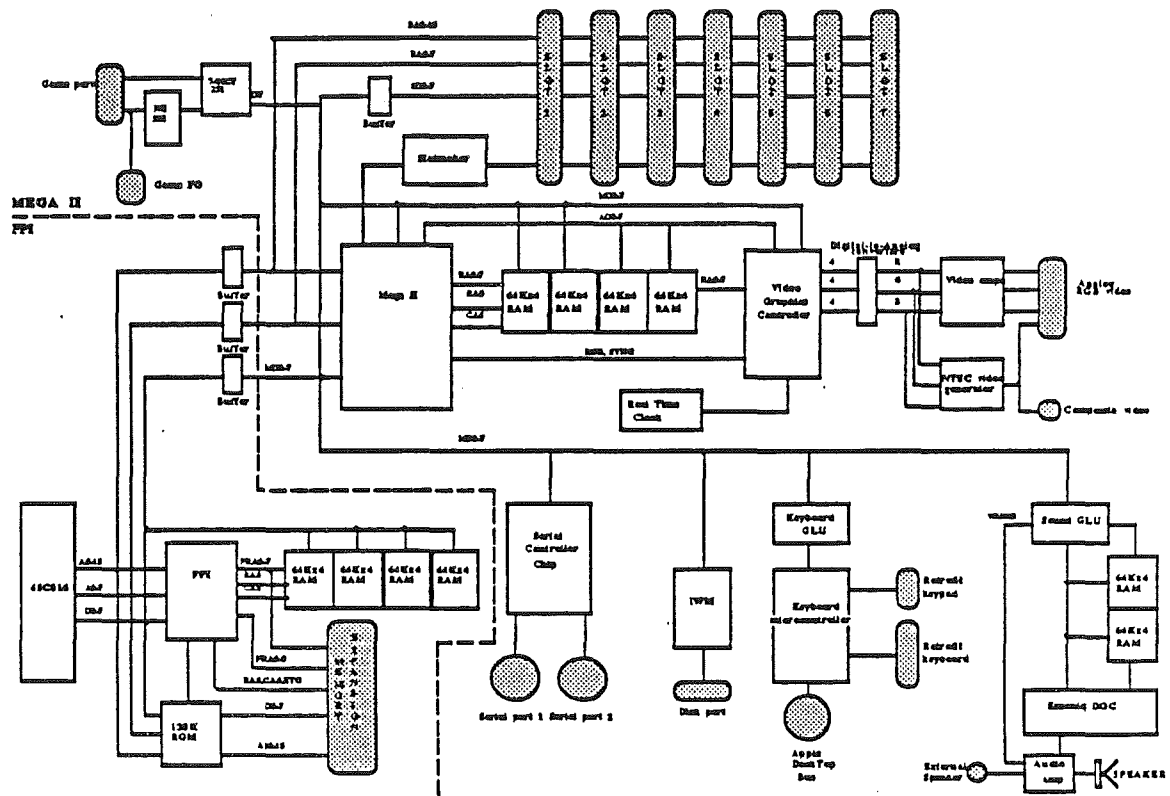


Figure 1-6: Cortland system block diagram.

The rest of this chapter describes the FPI and Mega II subsystems in more detail and how they function together.

Apple II Compatibility

The Cortland is compatible with the Apple II family of processors. Here are some of the features the Cortland shares with the Apple //e and //c:

- 6502 processor compatibility which is maintained by the 65816 microprocessor used by the Cortland

Cortland Hardware Reference Manual

- **Apple II graphics** which includes Lo-Res mode, Hi-Res mode and double Hi-Res mode color video graphics
- 128 K of main RAM memory
- Built-in Applesoft BASIC
- Two built-in Serial ports like the //c
- Seven peripheral expansion slots, compatible with the //e
- A built-in disk interface port that will accept either 5.25-inch or 3.5-inch disk drives
- Built-in Apple II monitor firmware
- 40-column and 80-column text display capability
- A game I/O port for joysticks and paddles like the //e

New Cortland features

The while the Cortland has many features in common with previous Apple II products, it has several new features added to enhance it's performance. Here are a few examples:

- **The sixteen-bit CMOS 65816** microprocessor which uses a superset of the 6502 instruction set and which includes eleven new address modes and 36 new instructions. The 65816 is completely compatible with 6502 and 65C02 code;

Sidebar:

Complimentary Metal Oxide Semiconductor (CMOS) is the silicon material that this microprocessor is made from. Using this material allows the manufacturer to make the device faster and require less power.

Sidebar:

To learn more about the 6502 and the 65C02 microprocessors, refer to the *Apple IIe Technical Reference Manual* and the *Apple IIc Technical Reference Manual*, respectively.

- **High processing speed** which is selectable between 1.024 MHz or 2.8 MHz
- **Super Hi-Res video graphics** mode which offers either 320- or 640-pixel horizontal resolution, displaying either 16 or 4 colors per line, respectively. These colors may be chosen from a total of 4096 possible colors.
- **Analog RGB color video outputs**, which are NTSC-compatible

- **128K bytes of RAM** which may be expanded to a maximum of 8 megabytes which can be achieved by using an optional expansion card in the memory expansion slot. A maximum of one megabyte of ROM can be utilized by using an expansion card.
- **Built-in AppleTalk network firmware**
- **Built-in real-time clock (RTC)** with a back-up battery, which is accessible through the control panel
- **A color border and color text** which are available at the RGB outputs
- **A sound synthesizer IC** with 15 voices
- **A detachable, full international keyboard** with keypad
- **Apple DeskTop bus** whose protocol provides for input devices such as graphics pads, mice and keyboards
- **Enhanced monitor firmware** which supports the 65816 microprocessor
- **A control panel screen** which provides users with means for setting system parameters;

Terminology

The terms "Mega II subsystem" and "Apple II-side" refer to the portion of the Cortland which provides the Apple II compatibility. The terms "FPI subsystem" and "16-bit side" refer to the portion of the Cortland which provides those features which are new to the Apple II family: everything that makes the Cortland not an Apple //e or //c.

Memory allocation

Note that the Cortland has three separate quantities of RAM: 128K available to the Mega II; 128K available to the FPI; 64K dedicated to the DOC. Figure 1-7 shows the Cortland memory map.

\$0	64K bytes	On-board FPI 128K RAM and I/O
\$1	64K bytes	
\$02-7F	8M bytes	Memory expansion card, RAM area
\$80-DF	6M bytes	Reserved - currently unused
\$E0	64K bytes	On-board Mega II 128K RAM
\$E1	64K bytes	
\$E2-EF	896K bytes	Reserved - currently unused
\$F0-FD	896K bytes	Memory expansion card, ROM area
\$FE-FF	64/128K bytes	Main board fast ROM area

Figure 1-7. Bank Memory Map

A minimum Cortland system includes 256K of RAM and 128K bytes of ROM. The Mega II subsystem has access to 128K of RAM which corresponds to the Main 64K and Auxiliary 64K memory banks of the Apple //e and Apple //c. Note that these banks are the total memory available to the Apple II-side of the system. The FPI may access 128K of RAM, and is expandible to 8 megabytes. The 128K of ROM is expandible to 1 megabyte.

Summary

The Cortland is a new Apple II product which carries over previous designs from the Apple //e and //c, yet also introduces some new features:

- New color graphics
- Synthesized sound capability
- New input-device bus
- Built-in clock

Chapter 2

The Mega II: Maintaining Compatibility

Introduction

The Cortland maintains compatibility with the rest of the Apple II computers by virtue of the Mega II custom IC. This chip contains most of the components from the Apple IIe (and many from the IIc) and is responsible for making it possible for the Cortland to run application programs written for the IIe.

This chapter describes the function of the Mega II IC and therefore the Apple IIe. Although each topic (video, I/O, memory, etc.) may also be found elsewhere in this manual, only those standard Apple II-compatible features are covered in this chapter. To read about the new features of the Cortland, see the chapter for each topic.

It contains detailed descriptions of all of the hardware and firmware that make up the Apple IIe, and therefore, all that makes the Cortland compatible with the Apple IIe.

This chapter contains a lot of information about the way the Cortland works, but it doesn't tell you how to use an Cortland. For this, you should read the other Cortland manuals, especially the *Cortland Owner's Guide*.

To read about the Cortland's new sound synthesizing capability, see *Chapter 6: Cortland Sound*.

To read about the Cortland's new Super Hi-Res graphics, see *Chapter 4: Video*.

Feature	Cortland	Apple IIc	Apple IIgs
Memory	64K	128K	128K
Serial Ports	expansion card only	2 built-in	2 built-in
disk port	expansion card only	1 built-in	1 built-in
text display	40-column (80 optional)	40- and 80-column	40- and 80-column
graphics	all Apple II graphics modes	all Apple II graphics modes	all Apple II graphics modes
keyboard device	built-in	built-in	detachable input-bus

Table 2-1. Apple II-compatible features.

Mega II Custom IC

The Mega II is a custom integrated circuit that is made up of several circuits previously found in the Apple //e. The following comprise the Mega II:

- Memory Management Unit (MMU) Custom IC
- Input Output Unit (IOU) Custom IC
- Character Generator ROMs (8 character sets)
- General Logic Unit (GLU) Custom IC
- Video Circuitry

The Mega II has virtually all of the functions of an Apple //e on a chip; it supports a slotted microcomputer architecture as well as the new peripherals built into the Cortland. Exceptions to this are the microprocessor, RAM and ROM memory, the slots, and the 16-pin Game I/O connector. Figure 2-1 shows the Mega II and its relationship within the Cortland.

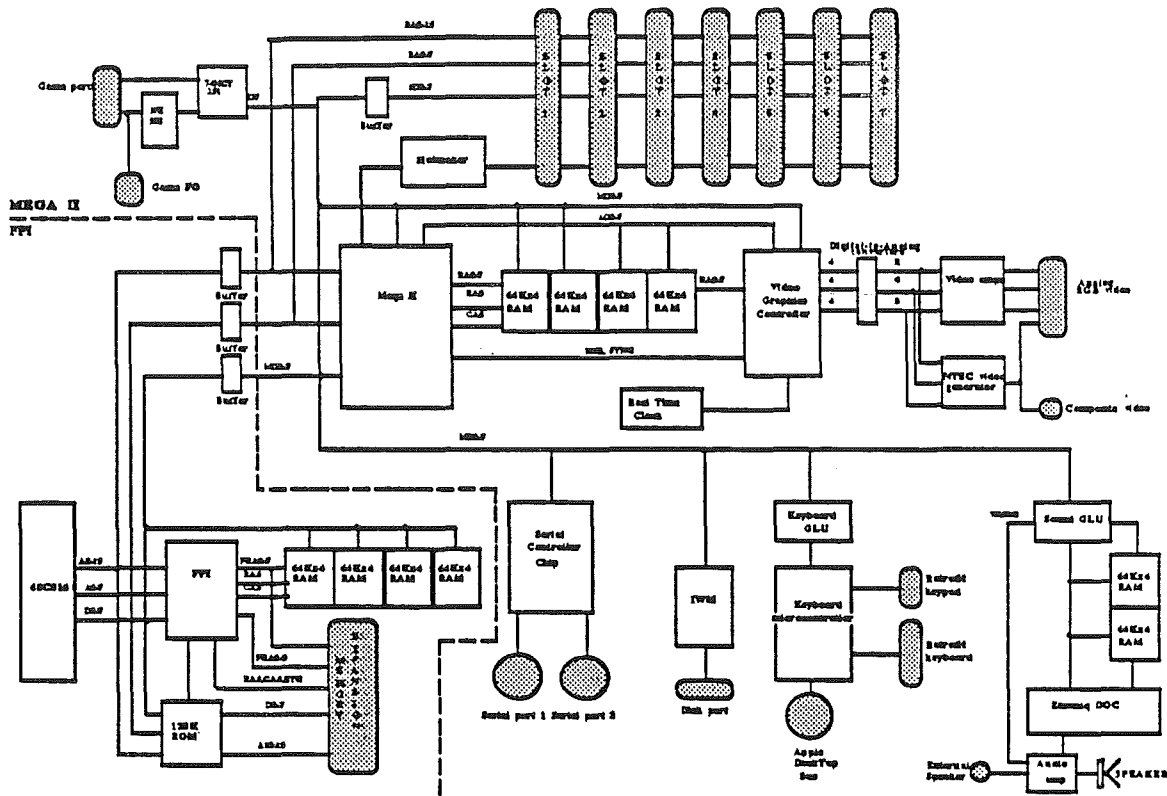


Figure 2-1. The Cortland block diagram.

The Apple IIe

The Cortland emulates many features of the Apple IIe:

- the 6502 microprocessor running at 1.024 MHz
- 128K of RAM
- 40-column display and 80-column text display
- Video graphics including:
 - Lo-Res graphics
 - Hi-Res graphics
 - Double Hi-Res graphics
- Peripheral card I/O including DMA

Although the Cortland is capable of running the processor at a higher clock speed, many standard Apple II application programs must be executed at the slower speed for timing reasons.

RAM upgrade

The original Apple IIe is a 64K machine, expandable to 128K through the use of auxiliary memory cards like the Extended

Cortland Hardware Reference Manual

80-Column Text Card. The Cortland has 128K of main memory, mounted on the circuit board.

The eight 64Kx1 RAM ICs on the IIe circuit board have been replaced by four 64Kx4 ICs on the Cortland circuit board. This means that the Apple Cortland has four RAM ICs instead of eight like the original IIe, while resulting in twice the resident memory.

Although the total memory installed in the Cortland may be several megabytes, only 128K is available to standard Apple II application programs in Apple II emulation mode.

Apple II I/O

This section describes the input and output (I/O) devices built into the Cortland in terms of their functions and the way they are used by programs. The built-in I/O devices are:

- the keyboard
- the video-display generator
- the speaker
- the game input and output

At the lowest level, programs use the built-in I/O devices by reading and writing to dedicated memory locations. This section lists these locations for each I/O device. It also gives the locations of the internal soft-switches that select the different display modes of the Cortland.

- ◆ *Built-in I/O routines:* This method of input and output—loading and storing directly to specific locations in memory—is not the only method you can use. For many of your programs, it may be more convenient to call the built-in I/O routines stored in the Cortland's firmware.

To read more about Built-In I/O routines in the Cortland, see the *Cortland Firmware Reference manual*.

The keyboard

The keyboard is the primary input device using the Apple DeskTop Bus (ADB) to communicate with the processor. All input devices are connected to the ADB and are controlled by the Keyboard Microcontroller. This controller supports reading of the keyboard by standard Apple II application programs.

The Cortland keyboard has automatic repeat, which means that if you press any key longer than you would during normal typing, the character code for that key will be sent continuously until you release the key. This also allow you to

hold down any number of keys and still press another key; this is known as N-key rollover.

Cortland's manufactured for sale outside the United States have a slightly different standard keyboard arrangement and include provisions for switching between different character sets. These differences are described in Appendix I.

Reading the keyboard

The keyboard encoder and ROM generate all 128 ASCII codes, so all of the special character codes in the ASCII character set are available from the keyboard. Application programs obtain character codes from the keyboard by reading a byte from the keyboard-data location shown in Table 2-1.

Table 2-1
Keyboard memory locations

Hex	Location		Description
	Decimal		
\$C000	49152	-16384	Keyboard data and strobe
\$C010	49168	-16368	Any-key-down flag and clear-strobe switch

Your programs can get the code for the last key pressed by reading the keyboard-data location. Table 2-1 gives this location in three different forms: the hexadecimal value used in assembly language, indicated by a preceding dollar sign (\$); the decimal value is used in Applesoft BASIC. The low-order seven bits of the byte at the keyboard location contain the character code; the high-order bit of this byte is the strobe bit, described below.

Your program can find out whether any key is down, except the Reset, Control, Shift, Caps Lock, ⌘ and Option keys, by reading from location 49152 (hex \$C000). The high-order bit (bit 7) of the byte you read at this location is called any-key-down; it is 1 if a key is down, and 0 if no key is down. The value of this bit is 128; if a BASIC program gets this information with a PEEK, the value is 128 or greater if any key is down, and less than 128 if no key is down.

The ⌘ and Option keys are connected to switches 0 and 1 of the game I/O connector inputs. If OA is pressed, switch 0 is "pressed," and if the Option key is pressed, switch 1 is "pressed."

The strobe bit is the high-order bit of the keyboard-data byte. After any key has been pressed, the strobe bit is high. It remains high until you reset it by reading or writing at the clear-strobe location. This location is a combination flag and switch; the flag tells whether any key is down, and the switch

clears the strobe bit. The switch function of this memory location is called a soft switch because it is controlled by software. In this case, it doesn't matter whether the program reads or writes, and it doesn't matter what data the program writes: the only action that occurs is the resetting of the keyboard strobe. Similar soft switches, described later, are used for controlling other functions in the Cortland.

Important Any time you read the any-key-down flag, you also clear the keyboard strobe. If your program needs to read both the flag and the strobe, it must read the strobe bit first.

After the keyboard strobe has been cleared, it remains low until another key is pressed. Even after you have cleared the strobe, you can still read the character code at the keyboard location. The data byte has a different value, because the high-order bit is no longer set, but the ASCII code in the seven low-order bits is the same until another key is pressed. Appendix C contains the ASCII codes for most of the keys on the keyboard of the Cortland.

There are several special-function keys that do not generate ASCII codes. For example, pressing the Control, Shift, or Caps Lock key directly alters the character codes produced by the other keys. In the Cortland, the state of these modifier keys is available by reading a register within the Keyboard Microcontroller.

To learn how to read the registers within the Keyboard Microcontroller, see *Chapter 7: Apple DeskTop Bus*.

The Reset key is different from all other keys on the ADB keyboard only in that it generates a special key code. When the Keyboard Microcontroller detects the reset code alone, it asserts the RESET line, and resets the system. If the Reset and Control keys are both depressed, the system halts whatever program it's running and restarts itself. This restarting process is called the *reset routine*.

To read about the reset routine, see the *Cortland Firmware Reference manual*.

To read more about communication on the ADB, see *Chapter 8: Apple DeskTop Bus*.

Video

The video display generator

The primary output device of the Cortland is the video display. You can use any ordinary video monitor, either

color or black-and-white, to display video information from the Cortland. An ordinary monitor is one that accepts composite video compatible with the standard set by the NTSC (National Television Standards Committee). If you use standard Apple II color graphics with a monochrome (single-color) monitor, the display will appear as that color (black, for example) and various patterns made up of shades of that color.

If you are using only 40-column text and graphics modes, you can use a television set for your video display. If the TV set has an input connector for composite video, you can connect it directly to your computer; if it does not, you'll need to attach a radio frequency (RF) video modulator between the Cortland and the television set.

Important The Cortland can produce an 80-column text display. However, if you use an ordinary color or black-and-white television set, 80-column text will be too blurry to read. For a clear 80-column display, you must use a high-resolution video monitor with a bandwidth of 14 MHz or greater.

The specifications for the video display are summarized in Table 2-2.

The video signal produced by the Cortland is NTSC-compatible composite color video. It is available at two places: the RCA-type phono jack and at the RGB video connector, both on the back of the Cortland. Use the RCA-type phono jack to connect a composite video monitor or an external video modulator; use the RGB video connector to connect an analog-type RGB monitor.

Table 2-2
Video display specifications

Display modes:	40-column text; map: Figure 2-6 80-column text; map: Figure 2-7 Low-resolution color graphics; map: Figure 2-8 High-resolution color graphics; map: Figure 2-9 Double-high-res color graphics; map: Figure 2-10
Text capacity:	24 lines by 80 columns (character positions)
Character set:	128 ASCII characters (see Appendix C for a list of display characters)
Display formats:	Normal, inverse, flashing, MouseText (Table 2-4)
Low-resolution graphics:	16 colors (Table 2-5) 40 horizontal by 48 vertical; map: Figure 2-8
High-resolution graphics:	6 colors (Table 2-6) 140 horizontal by 192 vertical (restricted) Black-and-white: 280 horizontal by 192 vertical; map: Figure 2-9

Cortland Hardware Reference Manual

Double -high-resolution graphics: 16 colors (Table 2-7) 140 horizontal by 192 vertical (no restrictions)
Black-and-white: 560 horizontal by 192 vertical; map: Figure 2-10

The Cortland can produce seven different kinds of standard Apple II video display:

- text, 24 lines of 40 characters
- text, 24 lines of 80 characters
- low-resolution graphics, 40 by 48, in 16 colors
- high-resolution graphics, 140 by 192, in 6 colors
- high-resolution graphics, 280 by 192, in black and white
- double high-resolution graphics, 140 by 192, in 16 colors
- double high-resolution graphics, 560 by 192, in black and white

The two text modes can display all 128 ASCII characters: uppercase and lowercase letters, numbers, and symbols. The Cortland can also display MouseText characters.

Any of the graphics displays can have 4 lines of text at the bottom of the screen. The text may be either 40-column or 80-column, except that double high-resolution graphics may only have 80-column text at the bottom of the screen. Graphics displays with text at the bottom are called mixed-mode displays.

The low-resolution graphics display is an array of colored blocks, 40 wide by 48 high, in any of 16 colors. In mixed mode, the 4 lines of text replace the bottom 8 rows of blocks, leaving 40 rows of 40 blocks each.

The high-resolution graphics display is an array of dots, 280 wide by 192 high. There are 6 colors available in high-resolution displays, but a given dot can use only 4 of the 6 colors. If color is used, the display is 140 dots wide by 192 high. In mixed mode, the 4 lines of text replace the bottom 32 rows of dots, leaving 160 rows of 280 dots each.

The double high-resolution graphics display uses main and auxiliary memory to display an array of dots, 560 wide by 192 high. All the dots are visible in black and white. If color is used, the display is 140 dots wide by 192 high with 16 colors available. In mixed mode, the 4 lines of text replace the bottom 32 rows of dots, leaving 160 rows of 560 (or 140) dots each. In mixed mode, the text lines can be 80 columns wide only.

The Cortland can also display Super Hi-Res graphics, although it is not a standard Apple II video display mode. To read about Super Hi-Res graphics, see *Chapter 4: Video*.

Text modes

The text characters displayed include the uppercase and lowercase letters, the ten digits, punctuation marks, and special characters. Each character is displayed in an area of the screen that is seven dots wide by eight dots high. The characters are formed by a dot matrix five dots wide, leaving two blank columns of dots between characters in a row, except for MouseText characters, some of which are seven dots wide. Except for lowercase letters with descenders and some MouseText characters, the characters are only seven dots high, leaving one blank line of dots between rows of characters.

The normal display has white (or other single color) dots on a black background. Characters can also be displayed as black dots on a white background; this is called *inverse format*.

Text character sets

The Cortland can display either of two text character sets: the primary set or an alternate set. The forms of the characters in the two sets are actually the same, but the available display formats are different. The display formats are

- normal, with white dots on a black screen
- inverse, with black dots on a white screen
- flashing, alternating between normal and inverse

With the primary character set, the Cortland can display uppercase characters in all three formats: normal, inverse, and flashing. Lowercase letters can only be displayed in normal format. The primary character set is compatible with most software written for other Apple II models, which can display text in flashing format but don't have lowercase characters.

The alternate character set displays characters in either normal or inverse format. In normal format, you can get

- uppercase letter
- lowercase letters
- numbers
- special characters

In inverse format, you can get

- MouseText characters
- uppercase letters
- lowercase letters
- numbers
- special characters.

You select the character set by means of the alternate-text soft switch, ALTCHAR, described later in the section "Display Mode Switching." Table 2-4 shows the character codes in hexadecimal for the primary and alternate character sets in normal, inverse, and flashing formats.

Each character on the screen is stored as one byte of display data. The low-order six bits make up the ASCII code of the character being displayed. The remaining two (high-order) bits select inverse or flashing format and uppercase or lowercase characters. In the primary character set, bit 7 selects inverse or normal format and bit 6 controls character flashing. In the alternate character set, bit 6 selects between uppercase and lowercase, according to the ASCII character codes, and flashing format is not available.

Table 2-3
Display character sets

Hex Values	Primary character set		Alternate character set	
	Character type	Format	Character type	Format
\$00-\$1F	Uppercase letters	Inverse	Uppercase letters	Inverse
\$20-\$3F	Special characters	Inverse	Special characters	Inverse
\$40-\$5F	Uppercase letters	Flashing	MouseText	Inverse
\$60-\$7F	Special characters	Flashing	Lowercase letters	Inverse
\$80-\$9F	Uppercase letters	Normal	Uppercase letters	Normal
\$A0-\$BF	Special characters	Normal	Special characters	Normal
\$C0-\$DF	Uppercase letters	Normal	Uppercase letters	Normal
\$E0-\$FF	Lowercase letters	Normal	Lowercase letters	Normal

40-column versus 80-column text

The Cortland has two modes of text display: 40-column and 80-column. The number of dots in each character does not change, but the characters in 80-column mode are only half as wide as the characters in 40-column mode. Compare Figure 2-3 and Figure 2-4. On an ordinary color or black-and-white television set, the narrow characters in the 80-column display blur together; you must use the 40-column mode to display text on a television set.

Figure 2-3
40-column text display

```
]LIST 0,100
10 REM APPLESOFT CHARACTER DEMO
20 TEXT : HOME
30 PRINT : PRINT "Applesoft character Demo"
40 PRINT : PRINT "Which character set--"
50 PRINT : INPUT "Primary (P) or Alternate (A) ?";AS
60 IF LEN (AS) < 1 THEN 50
65 LET AS = LEFT$ (AS,1)
70 IF AS = "P" THEN POKE 49166, 0
80 IF AS = "A" THEN POKE 49167, 0
90 PRINT : PRINT "...printing the same line, first"
100 PRINT " in NORMAL, then INVERSE ,then FLASH:": PRINT
]
```

Figure 2-4
80-column text display

```
]LIST
10 REM APPLESOFT CHARACTER DEMO
20 TEXT : HOME
30 PRINT : PRINT "Applesoft Character Demo"
40 PRINT : PRINT "Which character set--"
50 PRINT : INPUT "Primary (P) or Alternate (A) ?";AS
60 IF LEN (AS) < 1 THEN 50
65 LET AS = LEFT$ (AS,1)
70 IF AS = "P" THEN POKE 49166,0
80 IF AS = "A" THEN POKE 49167,0
90 PRINT : PRINT "printing the same line, first"
100 PRINT " in NORMAL, then INVERSE ,then FLASH:": PRINT
150 NORMAL : GOSUB 1000
160 INVERSE : GOSUB 1000
170 FLASH : GOSUB 1000
180 NORMAL : PRINT : PRINT : PRINT "Press any key to repeat."
190 GET AS
200 GOTO 10
1000 PRINT : PRINT "SAMPLE TEXT: Now is the time--12:00"
1100 RETURN
]
```

Graphics modes

The Cortland can produce video graphics in three different modes. All the graphics modes treat the screen as a rectangular array of spots. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes in these arrays; this section describes the way the resulting graphics data are stored in the Cortland's memory.

Low-resolution graphics

In the low-resolution graphics mode, the Cortland displays an array of 48 rows by 40 columns of colored blocks. Each block can be any one of sixteen colors, including black and white. On a black-and-white monitor or television set, these colors appear as black, white, and three shades of gray. There are no blank dots between blocks; adjacent blocks of the same color merge to make a larger shape.

Data for the low-resolution graphics display is stored in the same part of memory as the data for the 40-column text display. Each byte contains data for two low-resolution graphics blocks. The two blocks are displayed one atop the other in a display space the same size as a 40-column text character, seven dots wide by eight dots high.

Half a byte—four bits, or one nibble—is assigned to each graphics block. Each nibble can have a value from 0 to 15, and this value determines which one of sixteen colors appears on the screen. The colors and their corresponding nibble values are shown in

Table 2-4. In each byte, the low-order nibble sets the color for the top block of the pair, and the high-order nibble sets the color for the bottom block. Thus, a byte containing the hexadecimal value `$D8` produces a brown block atop a yellow block on the screen.

Table 2-4

Low-resolution graphics colors

Note: Colors may vary, depending upon the controls on the monitor or TV set.

Nibble value			Nibble value		
Dec	Hex	Color	Dec	Hex	Color
0	\$00	Black	8	\$08	Brown
1	\$01	Magenta	9	\$09	Orange
2	\$02	Dark blue	10	\$0A	Gray 2
3	\$03	Purple	11	\$0B	Pink
4	\$04	Dark green	12	\$0C	Light green
5	\$05	Gray 1	13	\$0D	Yellow
6	\$06	Medium Blue	14	\$0E	Aquamarine
7	\$07	Light blue	15	\$0F	White

As explained later in the section "Display Pages," the text display and the low-resolution graphics display use the same area in memory. Most programs that generate text and graphics clear this part of memory when they change display modes, but it is possible to store data as text and display it as graphics, or vice-versa. All you have to do is change the mode switch, described later in this chapter in the section "Display Mode Switching," without changing the display data. This usually produces meaningless jumbles on the display, but some programs have used this technique to good advantage for producing complex low-resolution graphics displays quickly.

High-resolution graphics

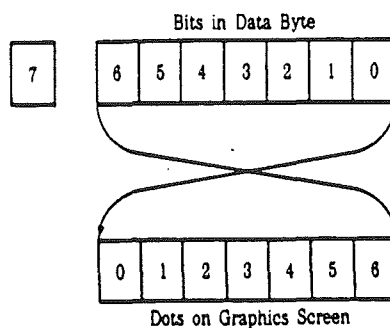
In the high-resolution graphics mode, the Cortland displays an array of colored dots in 192 rows and 280 columns. The colors available are black, white, purple, green, orange, and blue, although the colors of the individual dots are limited, as described later in this section. Adjacent dots of the same color merge to form a larger colored area.

Data for the high-resolution graphics displays are stored in either of two 8192-byte areas in memory. These areas are called *high-resolution Page 1* and *Page 2*, think of them as buffers where you can put data to be displayed. Normally, your programs will use the features of some high-level language to draw graphics dots, lines, and shapes to display; this section describes the way the resulting graphics data are stored in the Cortland's memory.

The Cortland high-resolution graphics display is bit-mapped: each dot on the screen corresponds to a bit in the Cortland's memory. The seven low-order bits of each display byte control a row of seven adjacent dots on the screen, and forty

adjacent bytes in memory control a row of 280 (7 times 40) dots. The least significant bit of each byte is displayed as the leftmost dot in a row of seven, followed by the second-least significant bit, and so on, as shown in Figure 2-5. The eighth bit (the most significant) of each byte is not displayed; it selects one of two color sets, as described later.

Figure 2-5
High-resolution display bits



On a black-and-white monitor, there is a simple correspondence between bits in memory and dots on the screen. A dot is white if the bit controlling it is on (1), and the dot is black if the bit is off (0). On a black-and-white television set, pairs of dots blur together; alternating black and white dots merge to a continuous grey.

On an NTSC color monitor or a color television set, a dot whose controlling bit is off (0) is black. If the bit is on, the dot will be white or a color, depending on its position, the dots on either side, and the setting of the high-order bit of the byte.

Call the left-most column of dots column zero, and assume (for the moment) that the high-order bits of all the data bytes are off (0). If the bits that control dots in even-numbered columns (0, 2, 4, and so forth) are on, the dots are purple; if the bits that control odd-numbered columns are on, the dots are green—but only if the dots on both sides of a given dot are black. If two adjacent dots are both on, they are both white.

You select the other two colors, blue and orange, by turning the high-order bit (bit 7) of a data byte on (1). The colored dots controlled by a byte with the high-order bit on are either blue or orange: the dots in even-numbered columns are blue, and the dots in odd-numbered columns are orange—again, only if the dots on both sides are black. Within each horizontal line of seven dots controlled by a single byte, you can have black, white, and one pair of colors. To change the color of any dot to one of the other pair of colors, you must change the high-order bit of its byte, which affects the colors of all seven dots controlled by the byte.

In other words, high-resolution graphics displayed on a color monitor or television set are made up of colored dots, according to the following rules:

- Dots in even columns can be black, purple, or blue.
- Dots in odd columns can be black, green, or orange.
- If adjacent dots in a row are both on, they are both white.
- The colors in each row of seven dots controlled by a single byte are either purple and green, or blue and orange, depending on whether the high-order bit is off (0) or on (1).

These rules are summarized in Table 2-5. The blacks and whites are numbered to remind you that the high-order bit is different.

Table 2-5

High-resolution graphics colors

Note: Colors may vary depending upon the controls on the monitor or television set.

Bits 0-6	Bit 7 off	Bit 7 on
Adjacent columns off	Black 1	Black 2
Even columns on	Purple	Blue
Odd columns on	Green	Orange
Adjacent columns on	White 1	White 2

The peculiar behavior of the high-resolution colors reflects the way NTSC color television works. The dots that make up the Cortland video signal are spaced to coincide with the frequency of the color subcarrier used in the NTSC system. Alternating black and white dots at this spacing cause a color monitor or TV set to produce color, but two or more white dots together do not. Effective horizontal resolution with color is 140 dots per line ((280 divided by two).

Double high-resolution graphics

In the double high-resolution graphics mode, the Cortland displays an array of colored dots 560 columns wide and 192 rows deep. There are 16 colors available for use with double high-resolution graphics (see Table 2-6)

Double high-resolution graphics is a bit-mapping of the low-order seven bits of the bytes in the main-memory and auxiliary-memory pages at \$2000-\$3FFF. The bytes in the main-memory and auxiliary-memory pages are interleaved in exactly the same manner as the characters in 80-column text: of each pair of identical addresses, the auxiliary-memory byte is displayed first, and the main-memory byte is displayed

second. Horizontal resolution is 560 dots when displayed on a monochrome monitor.

Unlike high-resolution color, double high-resolution color has no restrictions on which colors can be adjacent. Color is determined by any four adjacent dots along a line. Think of a four-dot-wide window moving across the screen: at any given time, the color displayed will correspond to the four-bit value from Table 2-6 that corresponds to the window's position (Figure 2-10). Effective horizontal resolution with color is 140 (560 divided by four) dots per line.

To use Table 2-6, divide the display column number by four, and use the remainder to find the correct column in the table: *ab0* is a byte residing in auxiliary memory corresponding to a remainder of zero (byte 0, 4, 8, and so on); *mb1* is a byte residing in main memory corresponding to a remainder of one (byte 1, 5, 9 and so on), and similarly for *ab3* and *mb4*.

Table 2-6
Double high-resolution graphics colors

Repeated Color pattern	ab0	mb1	ab2	mb3	bit
Black	\$00	\$00	\$00	\$00	0000
Magenta	\$08	\$11	\$22	\$44	0001
Brown	\$44	\$08	\$11	\$22	0010
Orange	\$4C	\$19	\$33	\$66	0011
Dark green	\$22	\$44	\$08	\$11	0100
Gray 1	\$2A	\$55	\$2A	\$55	0101
Green	\$66	\$4C	\$19	\$33	0110
Yellow	\$6E	\$5D	\$3B	\$77	0111
Dark blue	\$11	\$22	\$44	\$08	1000
Purple	\$19	\$33	\$66	\$4C	1001
Gray 2	\$55	\$2A	\$55	\$2A	1010
Pink	\$5D	\$3B	\$77	\$6E	1011
Medium blue	\$33	\$66	\$4C	\$19	1100
Light blue	\$3B	\$77	\$6E	\$5D	1101
Aqua	\$77	\$6E	\$5D	\$3B	1110
White	\$7F	\$7F	\$7F	\$7F	1111

Video display pages

The Cortland generates its video displays using data stored in specific areas in memory. These areas, called *display pages*, serve as buffers where your programs can put data to be displayed. Each byte in a display buffer controls an object at a certain location on the display. In text mode, the object is a single character; in low-resolution graphics, the object is two stacked colored blocks; and in high-resolution and double high-resolution modes, it is a line of seven adjacent dots.

The 40-column-text and low-resolution-graphics modes use two display pages of 1024 bytes each. These are called *text Page 1* and *text Page 2*, and they are located at 1024-2047 (hexadecimal \$0400-\$07FF) and 2048-3071 (\$0800-\$0BFF) in main memory. Normally, only Page 1 is used, but you can put text or graphics data into Page 2 and switch displays instantly. Either page can be displayed as 40-column text, low-resolution graphics, or mixed-mode (four rows of text at the bottom of a graphics display).

The 80-column text mode displays twice as much data as the 40-column mode—1920 bytes—but it cannot switch pages. The 80-column text display uses a combination page made up of text Page 1 in main memory plus another page in auxiliary memory located on the 80-column text card. This additional memory is *not* the same as text Page 2—in fact, it occupies the same address space as text Page 1, and there is a special soft switch that enables you to store data into it. (See the next section, “Display Mode Switching.”) The built-in firmware I/O routines, described in Chapter 3, take care of this extra addressing automatically; that is one reason to use those routines for all your normal text output.

The high-resolution graphics mode also has two display pages, but each page is 8192 bytes long. In the 40-column text and low-resolution graphics modes each byte controls a display area seven dots wide by eight dots high. In high-resolution graphics mode each byte controls an area seven dots wide by one dot high. Thus, a high-resolution display requires eight times as much data storage, as shown in Table 2-7.

The double high-resolution graphics mode uses high-resolution Page 1 in both main and auxiliary memory. Each byte in those pages of memory controls a display area seven dots wide by one dot high. This gives you 560 dots per line in black and white, and 140 dots per line in color. A double high-resolution display requires twice the total memory as high-resolution graphics, and 16 times as much as a low-resolution display.

Table 2-7
Video display page locations

Display mode	Display page	Lowest address		Highest address	
		Hex	Dec	Hex	Dec
40-column text,	1	\$0400	1024	\$07FF	2047
low-resolution graphics	2*	\$0800	2048	\$0BFF	3071
80-column text	1	\$0400	1024	\$07FF	2047
	2*	\$0800	2048	\$0BFF	3071
High-resolution	1	\$2000	8192	\$3FFF	16383
graphics	2	\$4000	16384	\$5FFF	24575
Double high-	1†	\$2000	8192	\$3FFF	16383
resolution graphics	2†	\$4000	16384	\$5FFF	24575

* This is not supported by firmware; for instructions on how to switch pages, refer to the next section "Display Mode Switching."

† See the section "Double High-Resolution Graphics," earlier in this chapter.

Display mode switching

You select the display mode that is appropriate for your application by reading or writing to a reserved memory location called a *soft switch*. In the Cortland, most soft switches have three memory locations reserved for them: one for turning the switch on, one for turning it off, and one for reading the current state of the switch.

Table 2-8 shows the reserved locations for the soft switches that control the display modes. For example, to switch from mixed-mode to full-screen graphics in an assembly-language program, you could use the instruction

```
STA    $C052
```

To do this in a BASIC program, you could use the instruction

```
POKE 49234,0
```

Some of the soft switches in Table 2-8 must be read, some must be written to, and for some you can use either action. When writing to a soft switch, it doesn't matter what value you write; the action occurs when you address the location, and the value is ignored.

Table 2-8

Display soft switches

Note: W means write anything to the location, R means read the location, R/W means read or write, and R7 means read the location and then check bit 7.

Name	Action	Hex	Function
ALTCHAR	W	\$C00E	Off: display text using primary character set
ALTCHAR	W	\$C00F	On: display text using alternate character set
RDALTCHAR	R7	\$C01E	Read ALTCHAR switch (1 = on)
80COL	W	\$C00C	Off: display 40 columns
80COL	W	\$C00D	On: display 80 columns
RD80COL	R7	\$C01F	Read 80COL switch (1 = on)
80STORE	W	\$C000	Off: cause PAGE2 on to select auxiliary RAM
80STORE	W	\$C001	On: allow PAGE2 to switch main RAM areas
RD80STORE	R7	\$C018	Read 80STORE switch (1 = on)
PAGE2	R/W	\$C054	Off: select Page 1
PAGE2	R/W	\$C055	On: select Page 2 or, if 80STORE on, Page 1 in auxiliary memory
RDPAGE2	R7	\$C01C	Read PAGE2 switch (1 = on)
TEXT	R/W	\$C050	Off: display graphics or, if MIXED on, mixed
TEXT	R/W	\$C051	On: display text
RDTEXT	R7	\$C01A	Read TEXT switch (1 = on)
MIXED	R/W	\$C052	Off: display only text or only graphics
MIXED	R/W	\$C053	On: if TEXT off, display text and graphics
RDMIXED	R7	\$C01B	Read MIXED switch (1 = on)
HIRES	R/W	\$C056	Off: if TEXT off, display low-resolution graphics
HIRES	R/W	\$C057	On: if TEXT off, display high-resolution or, if DHIRES on, double high-resolution graphics
RDHIRES	R7	\$C01D	Read HIRES switch (1 = on)
DHIRES (AN3)	R/W	\$C05E	On: if IOUDIS on, turn on double high-res.
DHIRES (AN3)	R/W	\$C05F	Off: if IOUDIS on, turn off double high-res.
VBL	R7	\$C019	Read vertical blanking

❖ *By the Way:* You may not need to deal with these functions by reading and writing directly to the memory locations in Table 2-8. Many of the functions shown here are selected automatically if you use the display routines in the various high-level languages on the Cortland.

Any time you read a soft switch, you get a byte of data. However, the only information the byte contains is the state of the switch, and this occupies only one bit—bit 7, the high-order bit. The other bits in the byte are unpredictable.

If you read a soft switch from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if the switch is on, the value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

Addressing display pages directly

Before you decide to use the display pages directly, consider the alternatives. Most high-level languages enable you to write statements that control the text and graphics displays. Similarly, if you are programming in assembly language, you may be able to use the display features of the built-in I/O firmware. You should store directly into display memory only if the existing programs can't meet your requirements.

The display memory maps are shown in Figures 2-6, 2-7, 2-8, 2-9, and 2-10. All of the different display modes use the same basic addressing scheme: characters or graphics bytes are stored as rows of 40 contiguous bytes, but the rows themselves are not stored at locations corresponding to their locations on the display. Instead, the display address is transformed so that three rows that are eight rows apart on the display are grouped together and stored in the first 120 locations of each block of 128 bytes (\$80 hexadecimal). By folding the display data into memory this way, the Cortland, like the Apple II, stores all 960 characters of displayed text within 1K bytes of memory.

The high-resolution graphics display is stored in much the same way as text, but there are eight times as many bytes to store, because eight rows of dots occupy the same space on the display as one row of characters. The subset consisting of all the first rows from the groups of eight is stored in the first 1024 bytes of the high-resolution display page. The subset consisting of all the second rows from the groups of eight is stored in the second 1024 bytes, and so on for a total of 8 times 1024, or 8192 bytes. In other words, each block of 1024 bytes in the high-resolution display page contains one row of dots out of every group of eight rows. The individual rows are stored in sets of three 40-byte rows, the same way as the text display.

All of the display modes except 80-column mode and double high-resolution graphics mode can use either of two display pages. The display maps show addresses for each mode's Page 1 only. To obtain addresses for text or low-resolution graphics Page 2, add 1024 (\$400); to obtain addresses for high-resolution Page 2, add 8192 (\$2000).

The 80-column display and double high-resolution graphics mode work a little differently. Half of the data is stored in the normal text Page-1 memory, and the other half is stored in memory on the 80-column text card using the same addresses. The display circuitry fetches bytes from these two memory

Cortland Hardware Reference Manual

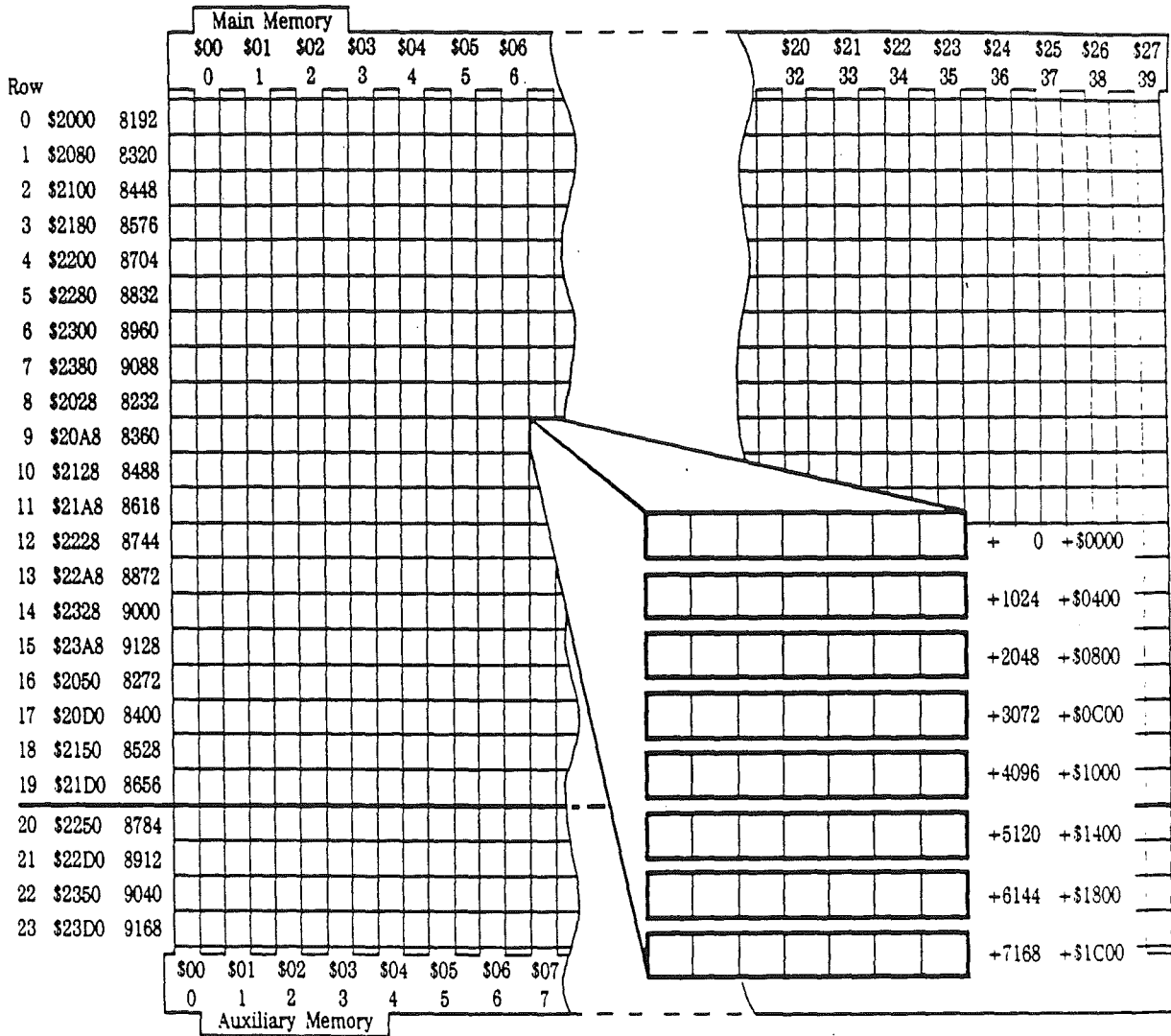
areas simultaneously and displays them sequentially: first the byte from the 80-column text display memory, then the byte from the main memory. The main memory stores the characters in the odd columns of the display, and the 80-column text display memory stores the characters in the even columns.

To store display data in the 80-column text display, first turn on the 80STORE soft switch by writing to location 49153 (hexadecimal \$C001 or complementary -16383). With 80STORE on, the page-select switch, PAGE2, selects between the portion of the 80-column display memory in Page 1 of main memory and the portion stored in the 80-column text display memory. To enable the 80-column text display, turn the PAGE2 soft switch on by reading or writing at location 49237.

Figure 2-8
Map of low-resolution graphics display

Row	0 \$00	1 \$01	2 \$02	3 \$03	4 \$04	5 \$05	6 \$06	7 \$07	8 \$08	9 \$09	10 \$0A	11 \$0B	12 \$0C	13 \$0D	14 \$0E	15 \$0F	16 \$10	17 \$11	18 \$12	19 \$13	20 \$14	21 \$15	22 \$16	23 \$17	24 \$18	25 \$19	26 \$1A	27 \$1B	28 \$1C	29 \$1D	30 \$1E	31 \$1F	32 \$20	33 \$21	34 \$22	35 \$23	36 \$24	37 \$25	38 \$26	39 \$27				
0	\$400	1024																																										
2	\$480	1152																																										
4	\$500	1280																																										
6	\$580	1408																																										
8	\$600	1536																																										
10	\$680	1664																																										
12	\$700	1792																																										
14	\$780	1920																																										
16	\$428	1064																																										
18	\$4A8	1192																																										
20	\$528	1320																																										
22	\$5A8	1448																																										
24	\$628	1576																																										
26	\$6A8	1704																																										
28	\$728	1832																																										
30	\$7A8	1960																																										
32	\$450	1104																																										
34	\$4D0	1232																																										
36	\$550	1360																																										
38	\$5D0	1488																																										
40	\$650	1616																																										
42	\$6D0	1744																																										
44	\$750	1872																																										
46	\$7D0	2000																																										

Figure 2-10
Map of double high-resolution graphics display



device or function requires a certain block of memory, there are more devices and functions than there are legal addresses, which means that the legal addresses must be shared. This sharing is accomplished through a technique called bank-switching, which is explained under the "Bank-Switched Memory" and "Auxiliary Memory and Firmware" sections in this chapter.

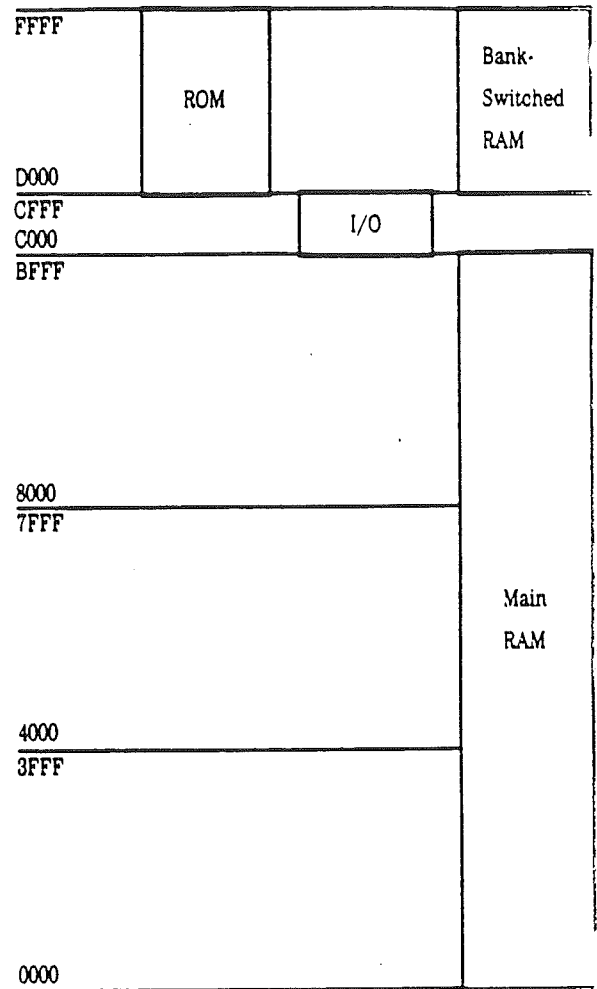
All input and output in the Cortland is *memory mapped*. This means that all devices connected to the Cortland appear to be a set of memory locations to the computer. In this chapter, the I/O memory spaces are described simply as blocks of memory.

Programmers often refer to the Cortland's memory in 256-byte blocks called *pages*. One reason for this is that a one-byte address counter or index register can specify one of 256 different locations. Thus, *page 0* consists of memory locations from 0 to 255 (hexadecimal \$00 to \$FF), inclusive; *page 1* consists of locations 256 to 511 (hexadecimal \$0100 to \$01FF). Note that the page number is the high-order part of the hexadecimal address. Don't confuse this kind of page with the display buffers in the Cortland, which are sometimes referred to as Page 1 and Page 2.

Main memory map

The map of the main memory address space in Figure 2-11 shows the functions of the major areas of memory.

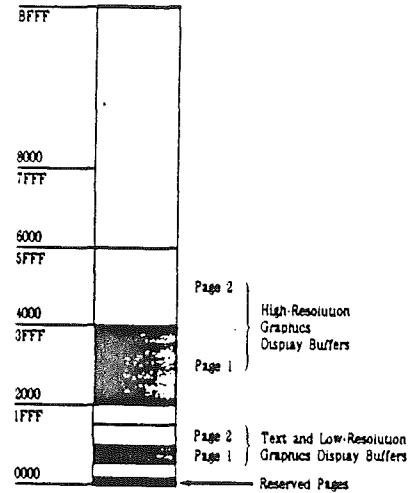
Figure 2-11
System memory map



RAM memory allocation

As Figure 2-11 shows, the major portion of the Cortland's memory space is allocated to program storage (RAM). Figure 2-12 shows the areas allocated to RAM. The main RAM memory extends from location 0 to location 49151 (hex \$BFFF), and occupies pages 0 through 191 (hex \$BF). There is also RAM storage in the bank-switched space from 53248 to 65535 (hex \$D000 to \$FFFF), described in the section "Bank-Switched Memory" later in this chapter, and auxiliary RAM, described in the section "Auxiliary Memory" later in this chapter.

Figure 2-12
RAM allocation map



Reserved memory pages

Most of the Cortland's RAM is available for storing your programs and data. However, a few RAM pages are reserved for the use of the Monitor firmware and the BASIC interpreters. The reserved pages are described in the following sections.

Important The system does not prevent your using these pages, but if you do use them, you must be careful not to disturb the system data they contain, or you will cause the system to malfunction.

Page zero

Several of the 65C02 microprocessor's addressing modes require the use of addresses in *page zero*, also called *zero page*. The Monitor, the BASIC interpreters, DOS 3.3, and ProDOS all make extensive use of page zero.

To use indirect addressing in your assembly-language programs, you must store base addresses in page zero. At the same time, you must avoid interfering with the other programs that use page zero—the Monitor, the BASIC interpreters, and the disk operating systems. One way to avoid conflicts is to use only those page-zero locations not already used by other programs.

The 65C02 stack

The 65C02 microprocessor uses page 1 as the *stack*—the place where subroutine return addresses are stored—in last-in, first-out sequence. Many programs also use the stack for temporary storage of the registers (via push and pull operations). You can do the same, but you should use it

sparingly. The stack pointer is eight bits long, so the stack can hold only 256 bytes of information at a time. When you store the 257th byte in the stack, the stack pointer repeats itself, or wraps around, so that the new byte replaces the first byte stored, which is now lost. This writing over old data is called *stack overflow*, and when it happens, the program continues to run normally until the lost information is needed, whereupon the program terminates catastrophically.

The input buffer

The GETLN input routine, which is used by the Monitor and the BASIC interpreters, uses page 2 as its keyboard-input buffer. The size of this buffer sets the maximum size of input strings. (Note: Applesoft uses only the first 237 bytes, although it permits you to type in 256 characters.) If you know that you won't be typing any long input strings, you can store temporary data at the upper end of page 2.

Link-address storage

The Monitor, ProDOS, and DOS 3.3 all use the upper part of page 3 for link addresses or vectors.

BASIC programs sometimes need short machine-language routines. These routines are usually stored in the lower part of page 3.

The display buffers

The primary text and low-resolution-graphics display buffer occupies memory pages 4 through 7 (locations 1024 through 2047, hexadecimal \$0400 through \$07FF). This entire 1024-byte area is called *text Page 1*, and it is not usable for program and data storage. There are 64 locations in this area that are not displayed on the screen; these locations are reserved for use by the peripheral cards.

Text Page 2, the alternate text and low-resolution-graphics display buffer, occupies memory pages 8 through 11 (locations 2048 through 3071, hexadecimal \$0800 through \$0BFF). Most programs do not use Page 2 for displays, so they can use this area for program or data storage.

The primary high-resolution-graphics display buffer, called *high-resolution Page 1*, occupies memory pages 32 through 63 (locations 8192 through 16383, hexadecimal \$2000 through \$3FFF). If your program doesn't use high-resolution graphics, this area is usable for programs or data.

High-resolution Page 2 occupies memory pages 64 through 95 (locations 16384 through 24575, hexadecimal \$4000 through

\$5FFF). Most programs use this area for program or data storage.

The primary double high-resolution-graphics display buffer, called *double high-resolution Page 1*, occupies memory pages 32 through 63 (locations 8192 through 16383, hexadecimal \$2000 through \$3FFF) in both main and auxiliary memory. If your program doesn't use high-resolution or double high-resolution graphics, this area of main memory is usable for programs or data.

Bank-switched memory

The memory address space from 52K to 64K (hexadecimal \$D000 through \$FFFF) is doubly allocated: it is used for both ROM and RAM. The 12K bytes of ROM (read-only memory) in this address space contain the Monitor and the Applesoft BASIC interpreter. Alternatively, there are 16K bytes of RAM in this space. The RAM is normally used for storing either the Integer BASIC interpreter or part of the Pascal Operating System (purchased separately).

You may be wondering why this part of memory has such a split personality. Some of the reasons are historical: the Cortland is able to run software written for a standard Apple II because it uses this part of memory in the same way they do. It's convenient to have the Applesoft interpreter in ROM, but the Cortland, like an Apple II with a language card, is also able to use that address space for other things when Applesoft is not needed.

You may also be wondering how 16K bytes of RAM are mapped into only 12K bytes of address space. The usual answer is that it's done with mirrors, and that isn't a bad analogy: the 4K-byte address space from 52K to 56K (hexadecimal \$D000 through \$DFFF) is used twice.

Switching different blocks of memory into the same address space is called *bank switching*. There are actually two examples of bank switching going on here: first, the entire address space from 52K to 64K (\$D000 through \$FFFF) is switched between ROM and RAM, and second, the address space from 52K to 56K (\$D000 to \$DFFF) is switched between two different blocks of RAM.

Figure 2-13
Bank-switched memory map

FFFF	ROM			RAM
E000				
DFFF			RAM	RAM
D000				

Setting bank switches

You switch banks of memory in the same way you switch other functions in a standard Apple II: by using soft switches. Read operations to these soft switches do three things: select either RAM or ROM in this memory space; enable or inhibit writing to the RAM; and select the first or second 4K-byte bank of RAM in the address space \$D000 to \$DFFF.

Warning Do not use these switches without careful planning. Careless switching between RAM and ROM is almost certain to have catastrophic effects on your program.

Table 2-12 shows the addresses of the soft switches for enabling all combinations of reading and writing in this memory space. All of the hexadecimal values of the addresses are of the form \$C08x. Notice that several addresses perform the same function: this is because the functions are activated by single address bits. For example, any address of the form \$C08x with a 1 in the low-order bit enables the RAM for writing. Similarly, bit 3 of the address selects which 4K block of RAM to use for the address space \$D000-\$DFFF; if bit 3 is 0, the first bank of RAM is used, and if bit 3 is 1, the second bank is used.

When RAM is not enabled for reading, the ROM in this address space is enabled. Even when RAM is not enabled for reading, it can still be written to if it is write-enabled.

When you turn power on or reset the Cortland, it initializes the bank switches for reading the ROM and writing the RAM, using the second bank of RAM. Note that this is different from the reset on the Apple II Plus, which didn't affect the bank-switched memory (the language card). On the Cortland, you can't use the reset vector to return control to a program in bank-switched memory, as you could on the Apple II Plus.

Table 2-12

Bank select switches

Note: R means read the location, W means write anything to the location, R/W means read or write, and R7 means read the location and then check bit 7.

Name	Action	Hex	Function
	R	\$C080	Read RAM; no write; use \$D000 bank 2.
	RR	\$C081	Read ROM; write RAM; use \$D000 bank 2.
	R	\$C082	Read ROM; no write; use \$D000 bank 2.
	RR	\$C083	Read and write RAM; use \$D000 bank 2.
	R	\$C088	Read RAM; no write; use \$D000 bank 1.
	RR	\$C089	Read ROM; write RAM; use \$D000 bank 1.
	R	\$C08A	Read ROM; no write; use \$D000 bank 1.
	RR	\$C08B	Read and write RAM; use \$D000 bank 1.
RDBNK2 R7		\$C011	Read whether \$D000 bank 2 (1) or bank 1 (0).
RDLGRAM	R7	\$C012	Reading RAM (1) or ROM (0).
ALTZP	W	\$C008	Off: use main bank, page 0 and page 1.
ALTZP	W	\$C009	On: use auxiliary bank, page 0 and page 1.
RDALTZP	R7	\$C016	Read whether auxiliary (1) or main (0) bank.

❖ *Reading and writing to RAM banks:* You can't read one RAM bank and write to the other; if you select either RAM bank for reading, you get that one for writing as well.

❖ *Reading RAM and ROM:* You can't read from ROM in part of the bank-switched memory and read from RAM in the rest: specifically, you can't read the Monitor in ROM while reading bank-switched RAM. If you want to use the Monitor firmware with a program in bank-switched RAM, copy the Monitor from ROM (locations \$F800 through \$FFCB) into bank-switched RAM. You can't do this from Pascal or ProDOS.

To see how to use these switches, look at the following section of an assembly-language program:

Cortland Hardware Reference Manual

AD 83 C0	LDA \$C083	*SELECT 2ND 4K BANK & READ/WRITE
AD 83 C0	LDA \$C083	*BY TWO CONSECUTIVE READS
A9 D0	LDA #\$D0	*SET UP...
85 01	STA BEGIN	*...NEW...
A9 FF	LDA #\$FF	*...MAIN-MEMORY...
85 02	STA END	*...POINTERS...
20 97 C9	JSR RAMTST	*...FOR 12K BANK
AD 8B C0	LDA \$C08B	*SELECT 1ST 4K BANK
20 97 C9	JSR RAMTST	*USE ABOVE POINTERS
AD 83 C0	LDA \$C088	*SELECT 1ST BANK & WRITE PROTECT
A9 80	LDA #\$80	
E6 10	INC TSTNUM	
20 58 C9	JSR WPTSINIT	
AD 80 C0	LDA \$C080	*SELECT 2ND BANK & WRITE PROTECT
E6 10	INC TSTNUM	
A9 01	LDA #PAT12K	
20 58 C9	JSR WPTSINIT	
AD 8B C0	LDA \$C08B	*SELECT 1ST BANK & READ/WRITE
AD 8B C0	LDA \$C08B	*BY TWO CONSECUTIVE READS
E6 0E	INC RWMODE	*FLAG RAM IN READ/WRITE
E6 10	INC TSTNUM	
A9 08	LDA #PAT4K	
20 58 C9	JSR WPTSINIT	

The LDA instruction, which performs a read operation to the specified memory location, is used for setting the soft switches. The unusual sequence of two consecutive LDA instructions performs the two consecutive reads that write-enable this area of RAM; in this case, the data that are read are not used.

Reading bank switches

You can read which language card bank is currently switched in by reading the soft switch at \$C011. You can find out whether the language card or ROM is switched in by reading \$C012. The only way that you can find out whether the language card RAM is write-enabled or not is by trying to write some data to the card's RAM space.

The State register

The State register is a read/write register containing eight of the standard Apple II soft switches. The byte-wide format of the soft switch state register simplifies the process of interrupt handling. Reading and storing this byte before executing interrupt routines allows the programmer to restore the system soft switches to the previous state in minimum time after returning from the interrupt routine.

STATE REGISTER FIGURE

Figure 2- . State register at \$C068.

Bit	Value	Description
7	-	ALZP - selects the alternate zero page
6	-	PAGE2 - selects the alternate text page
5	-	RAMRD - RAM read flag
4	-	RAMWRT - RAM write flag
3	-	RDROM - ROM read flag
2	-	BANK2 - selects the auxiliary RAM bank
1	-	ROMBANK - selects the ROM bank
0	-	SLOTcxROM - selects slot x ROM

Auxiliary memory

The auxiliary bank has a 1K-byte area of memory that serves the purpose of expanding the text display to 80 columns. The other 63K bytes can be used as auxiliary program and data storage. If you use only 40-column displays, the entire 64K bytes is available for programs and data.

Warning Do not attempt to use the auxiliary memory from a BASIC program. The BASIC Interpreter uses several areas in main

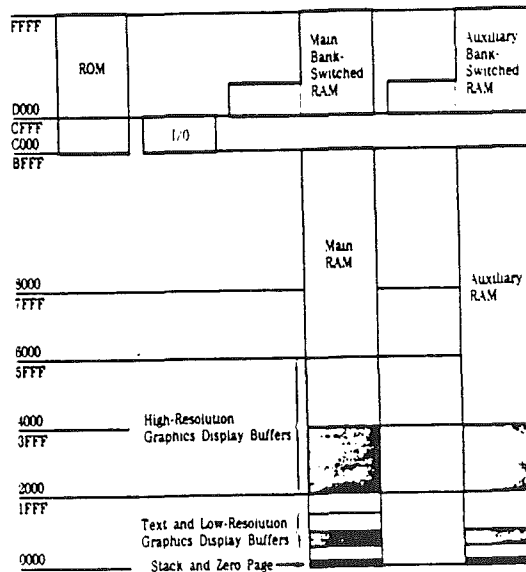
RAM, including the stack and the zero page. If you switch to auxiliary memory in these areas, the BASIC interpreter fails and you must reset the system and start over.

As you can see by studying the memory map in Figure 2-14, the auxiliary memory is broken into two large sections and one small one. The largest section is switched into the memory address space from 512 to 49151 (\$0200 through \$BFFF). This space includes the display buffer pages: as described in the section "Text Modes" in Chapter 2, space in auxiliary memory is used for one half of the 80-column text display. You can switch to the auxiliary memory for this entire memory space, or you can switch just the display pages: see the next section, "Memory Mode Switching."

- ◆ **Soft switches:** If the only reason you are using auxiliary memory is for the 80-column display, note that you can store into the display page in auxiliary memory by using the 80STORE and PAGE2 soft switches described in the section "Display Mode Switching" in Chapter 2.

The other large section of auxiliary memory is switched into the memory address space from 52K to 64K (\$D000 through \$FFFF). This memory space and the switches that control it are described earlier in this chapter in the section "Bank-Switched Memory." If you use the auxiliary RAM in this space, the soft switches have the same effect on the auxiliary RAM that they do on the main RAM: the bank switching is independent of the auxiliary-RAM switching.

Figure 2-14
Memory map with auxiliary memory



- ❖ **Bank switches:** Note that the soft switches for the bank-switched memory, described in the previous section, do not change when you switch to auxiliary RAM. In particular, if ROM is enabled in the bank-switched memory space before you switch to auxiliary memory, the ROM will still be enabled after you switch. Any time you switch the bank-switched section of auxiliary memory in and out, you must also make sure that the bank switches are set properly.

When you switch in the auxiliary RAM in the bank-switched space, you also switch the first two pages, from 0 to 511 (\$0000 through \$01FF). This part of memory contains page zero, which is used for important data and base addresses, and page one, which is the 65C02 stack. The stack and zero page are switched this way so that system software running in the bank-switched memory space can maintain its own stack and zero page while it manipulates the 48K address space (from \$0200 to \$BFFF) in either main memory or auxiliary memory.

Memory mode switching

Switching the 48K section of memory is performed by two soft switches: the switch named RAMRD selects main or auxiliary memory for reading, and the one named RAMWRT selects main or auxiliary memory for writing. As shown in Table 2-12, each switch has a pair of memory locations dedicated to it, one to select main memory, and the other to select auxiliary memory. Enabling the read and write functions independently makes it possible for a program whose instructions are being fetched from one memory space to store data into the other memory space.

Warning Do not use these switches without careful planning. Careless switching between main and auxiliary memories is almost certain to have catastrophic effects on the operation of the Cortland. For example, if you switch to auxiliary memory with no card in the slot, the program that is running will stop and you will have to reset the Cortland and start over.

Writing to the soft switch at location \$C003 turns RAMRD on and enables auxiliary memory for reading; writing to location \$C002 turns RAMRD off and enables main memory for reading. Writing to the soft switch at location \$C005 turns RAMWRT on and enables the auxiliary memory for writing; writing to location \$C004 turns RAMWRT off and enables main memory for writing. By setting these switches independently, you can use any of the four combinations of reading and writing in main or auxiliary memory.

Auxiliary memory corresponding to text Page 1 and high-resolution graphics Page 1 can be used as part of the address space from \$0200 to \$BFFF by using RAMRD and RAMWRT as

described above. These areas in auxiliary RAM can also be controlled separately by using the switches described in the section "Display Mode Switching" in Chapter 2. Those switches are named 80STORE, PAGE2, and HIRES.

As shown in Table 2-13, the 80STORE switch functions as an enabling switch: with it on, the PAGE2 switch selects main memory or auxiliary memory. With the HIRES switch off, the memory space switched by PAGE2 is the text Page 1, from \$0400 to \$07FF; with HIRES on, PAGE2 switches both text Page 1 and high-resolution graphics Page 1, from \$2000 to \$3FFF.

If you are using both the auxiliary-RAM control switches and the auxiliary-display-page control switches, the display-page control switches take priority: if 80STORE is off, RAMRD and RAMWRT work for the entire memory space from \$0200 to \$BFFF, but if 80STORE is on, RAMRD and RAMWRT have no effect on the display page. Specifically, if 80STORE is on and HIRES is off, PAGE2 controls text Page 1 regardless of the settings of RAMRD and RAMWRT. Likewise, if 80STORE and HIRES are both on, PAGE2 controls both text Page 1 and high-resolution graphics Page 1, again regardless of RAMRD and RAMWRT.

A single soft switch named ALTZP (for alternate zero page) switches the bank-switched memory and the associated stack and zero page area between main and auxiliary memory. As shown in Table 2-13, writing to location \$C009 turns ALTZP on and selects auxiliary-memory stack and zero page; writing to the soft switch at location \$C008 turns ALTZP off and selects main-memory stack and zero page for both reading and writing.

Table 2-13
Auxiliary-memory select switches

Name	Function	Hex	Location		Notes
				Decimal	
RAMRD	Read auxiliary memory	\$C003	49155	-16381	Write
	Read main memory	\$C002	49154	-16382	Write
	Read RAMRD switch	\$C013	49171	-16365	Read
RAMWRT	Write auxiliary memory	\$C005	49157	-16379	Write
	Write main memory	\$C004	49156	-16380	Write
	Read RAMWRT switch	\$C014	49172	-16354	Read
80STORE	On: access display page	\$C001	49153	-16383	Write
	Off: use RAMRD, RAMWRT	\$C000	49152	-16384	Write
	Read 80STORE switch	\$C018	49176	-16360	Read
PAGE2	Page 2 on (aux. memory)	\$C055	49237	-16299	*
	Page 2 off (main memory)	\$C054	49236	-16300	*
	Read PAGE2 switch	\$C01C	49180	-16356	Read
HIRES	On: access high-res pages	\$C057	49239	-16297	†
	Off: use RAMRD, RAMWRT	\$C056	49238	-16298	†
	Read HIRES switch	\$C01D	49181	-16355	Read
ALTZP	Aux. stack & zero page	\$C009	49161	-16373	Write
	Main stack & zero page	\$C008	49160	-16374	Write
	Read ALTZP switch	\$C016	49174	-16352	Read

* When 80STORE is on, the PAGE2 switch selects main or auxiliary display memory.

† When 80STORE is on, the HIRES switch enables you to use the PAGE2 switch to switch between the high-resolution Page 1 area in main memory or auxiliary memory.

There are three more locations associated with the auxiliary-memory switches. The high-order bits of the bytes you read at these locations tell you the settings of the three soft switches described above. The byte you read at location \$C013 has its high bit set to 1 if RAMRD is on (auxiliary memory is read-enabled), or 0 if RAMRD is off (the 48K block of main memory is read-enabled). The byte at location \$C014 has its high bit set to 1 if RAMWRT is on (auxiliary memory is write-enabled), or 0 if RAMWRT is off (the 48K block of main memory is write-enabled). The byte at location \$C016 has its high bit set to 1 if ALTZP is on (the bank-switched area, stack, and zero page in the auxiliary memory are selected), or 0 if ALTZP is off (these areas in main memory are selected).

♦ *Sharing memory:* In order to have enough memory locations for all of the soft switches and remain compatible with the Apple II and Apple II Plus, the soft switches listed in Table 2-13 share their memory locations with the keyboard functions listed in Table 2-1. The operations—read or write—shown in Table 2-13 for controlling the auxiliary memory are just the ones that are not used for reading the keyboard and clearing the strobe.

Peripheral Expansion

The seven expansion slots on the Cortland's main circuit board are used for installing circuit cards containing the hardware and firmware needed to interface peripheral devices to the Cortland. These slots are not simple I/O ports; peripheral cards can access the Cortland's data, address, and control lines via these slots. The expansion slots are numbered from 1 to 7, and certain signals, described below, are used to select a specific slot.

Interrupt support on the enhanced Cortland requires that special attention be paid to cards designed to be in slot 3. A description of what you need to watch for is given at the end of this chapter.

Selecting a device

The Apple Cortland supports several built-in devices and traditional slot-devices, with each device taking up one logical slot. Each built-in peripheral is assigned to a slot, and cards are plugged into any of the seven peripheral slots. This allows devices, such as a serial port, to be built onto the main logic board; however, only one device (either the built-in peripheral or the slot peripheral) can be selected at one time.

The Slot register

The control panel (accessible by pressing the Apple-Control-Reset keys simultaneously) allows the user to select the appropriate device for each logical slot. Enabling and disabling of internal peripherals may also be achieved under software control by setting the bits in the Slot Select register (location \$C02D). The bit representations are given here.

Bit	Value	Description
7	0	Selects the internal-device (AppleTalk) ROM code for slot 7 is selected.
	1	Enables both the slot-card ROM space (location \$C700 to \$C7FF) and I/O space \$C0F0 to \$C0FF.
6	0	Selects the internal-device (5-1/4" disk drive) ROM code for slot 6.
	1	Enables both the slot-card ROM space (location \$C600 to \$C6FF) and I/O space \$C0E0 to \$C0EF.
5	0	Selects the internal-device (3-1/2" disk drive) ROM code for slot 5.
	1	Enables both the slot-card ROM space (location \$C500 to \$C5FF) and I/O space \$C0D0 to \$C0DF.
4	0	Selects the internal-device (mouse) ROM code for slot 4.
	1	Enables both the slot-card ROM space (location \$C400 to \$C4FF).
3	-	Not used; must be set to zero. Slot 3 internal device (80-column circuitry) ROM code is always selected.

2	0	Selects the internal-device (serial port) ROM code for slot 2.
	1	Enables both the slot-card ROM space (location \$C200 to \$C2FF) and I/O space \$C0A0 to \$C0AF.
1	0	Selects the internal-device (serial port) ROM code for slot 1.
	1	Enables both the slot-card ROM space (location \$C100 to \$C1FF) and I/O space \$C090 to \$C09F.
0	-	Not used; must be set to zero.

Note: I/O space for slots 3 (\$C0C0 to \$C0CF) and 4 (\$C0D0 to \$C0DF) is always enabled.

Peripheral-card memory spaces

Because the Cortland's microprocessor does all of its I/O through memory locations, portions of the Cortland's memory space have been allocated for the exclusive use of the cards in the expansion slots. In addition to the memory locations used for actual I/O, there are memory spaces available for programmable memory (RAM) in the main memory and for read-only memory (ROM or PROM) on the peripheral cards themselves.

The memory spaces allocated for the peripheral cards are described below. Those memory spaces are used for small dedicated programs such as I/O drivers. Peripheral cards that contain their own driver routines in firmware like this are called intelligent peripherals. They make it possible for you to add peripheral hardware to your Cortland without having to change your programs, provided that your programs follow normal practice for data input and output.

Peripheral-card I/O space

Each expansion slot has the exclusive use of sixteen memory locations for data input and output in the memory space beginning at location \$C090. Slot 1 uses locations \$C090 through \$C09F, slot 2 uses locations \$C0A0 through \$C0AF, and so on through location \$C0FF, as shown in Table 2-14.

These memory locations are used for different I/O functions, depending on the design of each peripheral card. Whenever the Cortland addresses one of the sixteen I/O locations allocated to a particular slot, the signal on pin 41 of that slot, called DEVICE SELECT', switches to the active (low) state. This signal can be used to enable logic on the peripheral card that uses the four low-order address lines to determine which of its sixteen I/O locations is being accessed.

Table 2-14
Peripheral-card I/O memory locations enabled by DEVICE SELECT

Slot	Locations	Slot	Locations
1	\$C090-\$C09F	5	\$C0D0-\$C0DF
2	\$C0A0-\$C0AF	6	\$C0E0-\$C0EF
3	\$C0B0-\$C0BF	7	\$C0F0-\$C0FF
4	\$C0C0-\$C0CF		

Peripheral-card ROM space

One 256-byte page of memory space is allocated to each accessory card. This space is normally used for read-only memory (ROM or PROM) on the card with driver programs that control the operation of the peripheral device connected to the card.

The page of memory allocated to each expansion slot begins at location \$Cn00, where n is the slot number, as shown in Table 2-15 and Figure 2-13. Whenever the Cortland addresses one of the 256 ROM memory locations allocated to a particular slot, the signal on pin 1 of that slot, called I/O SELECT, switches to the active (low) state. This signal enables the ROM or PROM devices on the card, and the eight low-order address lines determine which of the 256 memory locations is being accessed.

Table 2-15
Peripheral-card I/O memory locations enabled by I/O SELECT

Locations	Slot	Locations
\$C100-\$C1FF	5	\$C500-\$C5FF
\$C200-\$C2FF	6	\$C600-\$C6FF
\$C300-\$C3FF	7	\$C700-\$C7FF
\$C400-\$C4FF		

Expansion ROM space

In addition to the small areas of ROM memory allocated to each expansion slot, peripheral cards can use the 2K-byte memory space from \$C800 to \$CFFF for larger programs in ROM or PROM. This memory space is called expansion ROM space. (See the memory map in Figure 2-3). Besides being larger, the expansion ROM memory space is always at the same locations regardless of which slot is occupied by the

card, making programs that occupy this memory space easier to write.

This memory space is available to any peripheral card that needs it. More than one peripheral card can have expansion ROM on it, but only one of them can be active at a time.

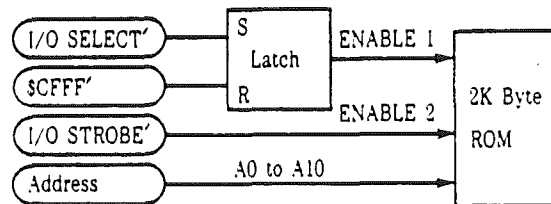
Each peripheral card that uses expansion ROM must have a circuit on it to enable the ROM. The circuit does this by a two-stage process: first, it sets a flip-flop when the I/O SELECT' signal, pin 1 on the slot, becomes active (low); second, it enables the expansion ROM devices when the I/O STROBE' signal, pin 20 on the slot, becomes active (low). Figure 2-15 shows a typical ROM-enable circuit.

The I/O SELECT' signal on a particular slot becomes active whenever the Cortland's microprocessor addresses a location in the 256-byte ROM address space allocated to that slot. The I/O STROBE' signal on all of the expansion slots becomes active (low) when the microprocessor addresses a location in the expansion-ROM memory space, \$C800-\$CFFF. The I/O STROBE' signal is used to enable the expansion-ROM devices on a peripheral card. (See Figure 2-15.)

Important

If there is an 80-column text card installed in the auxillary slot, some of the functions normally associated with slot 3 are performed by the 80-column text card and the built-in 80-column firmware. With the 80-column text card installed, the I/O STROBE' signal is not available on slot 3, so firmware in expansion ROM on a card in slot 3 will not run.

Figure 2-15
Expansion ROM enable circuit

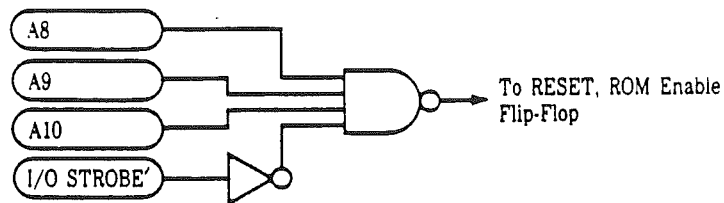


A program on a peripheral card can get exclusive use of the expansion ROM memory space by referring to location \$CFFF in its initialization phase. This location is special: all peripheral cards that use expansion ROM must recognize a reference to \$CFFF as a signal to disable their expansion ROMs. Of course, doing so also disables the expansion ROM on the card that is about to use it, but the next instruction in the initialization code sets the expansion-ROM enable circuit on the card.

A card that needs to use the expansion ROM space must first insert its slot address (\$Cn) in \$07F8 before it refers to \$CFFF.

This allows interrupting devices to reenable the card's expansion ROM after interrupt handling is finished. Once its slot address has been inserted in \$07F8, the peripheral card has exclusive use of the expansion memory space and its program can jump directly into the expansion ROM.

Figure 2-16
ROM disable address decoding



As described earlier, the expansion-ROM disable circuit resets the enable flip-flop whenever the 65C02 addresses location \$CFFF. To do this, the peripheral card must detect the presence of \$CFFF on the address bus. You can use the I/O STROBE' signal for part of the address decoding, since it is active for addresses from \$C800 through \$CFFF. If you can afford to sacrifice some ROM space, you can simplify the address decoding even further and save circuitry on the card. For example, if you give up the last 256 bytes of expansion ROM space, your disable circuit only needs to detect addresses of the form \$CFxx, and you can use the minimal disable-decoding circuitry shown in Figure 2-16.

Peripheral-card RAM space

There are 56 bytes of main memory allocated to the peripheral cards, eight bytes per card, as shown in Table 2-16. These 56 locations are actually in the RAM memory reserved for the text and low-resolution graphics displays, but these particular locations are not displayed on the screen and their contents are not changed by the built-in output routine COUT1. Programs in ROM on peripheral cards use these locations for temporary data storage.

Table 2-16
Peripheral-card RAM memory locations

Base address	Slot number						
	1	2	3*	4	5	6	7
\$0478	\$0479	\$047A	\$047B*	\$047C	\$047D	\$047E	\$047F
\$04F8	\$04F9	\$04FA	\$04FB*	\$04FC	\$04FD	\$04FE	\$04FF
\$0578	\$0579	\$057A	\$057B*	\$057C	\$057D	\$057E	\$057F
\$05F8	\$05F9	\$05FA	\$05FB*	\$05FC	\$05FD	\$05FE	\$05FF
\$0678	\$0679	\$067A	\$067B*	\$067C	\$067D	\$067E	\$067F
\$06F8	\$06F9	\$06FA	\$06FB*	\$06FC	\$06FD	\$06FE	\$06FF
\$0778	\$0779	\$077A	\$077B*	\$077C	\$077D	\$077E	\$077F
\$07F8	\$07F9	\$07FA	\$07FB*	\$07FC	\$07FD	\$07FE	\$07FF

* If there is a card in the auxiliary slot, it takes over these locations.

A program on a peripheral card can use the eight base addresses shown in the table to access the eight RAM locations allocated for its use, as shown in the next section, "I/O Programming Suggestions."

Warning The Cortland firmware always sets the value of \$04FB to \$FF on a reset.

I/O programming suggestions

A program in ROM on a peripheral card should work no matter which slot the card occupies. If the program includes a jump to an absolute location in one of the 256-byte memory spaces, then the card will work only when it is plugged into the slot that uses that memory space. If you are writing the program for a peripheral card that will be used by many people, you should avoid placing such a restriction on the use of the card.

Important To function properly no matter which slot a peripheral card is installed in, the program in the card's 256-byte memory space must not make any absolute references to itself. Instead of using jump instructions, you should force conditions on branch instructions, which use relative addressing.

The first thing a peripheral card used as an I/O device must do when called is to save the contents of the microprocessor's registers. (Peripheral cards not being used as I/O devices do not need to save the registers.) The device should save the register's contents on the stack, and restore them just before

returning control to the calling program. If there is RAM on the peripheral card, the information may be stored there.

Finding the slot number with ROM switched in

The memory addresses used by a program on a peripheral card differ depending on which expansion slot the card is installed in. Before it can refer to any of those addresses, the program must somehow determine the correct slot number. One way to do this is to execute a JSR (jump to subroutine) to a location with an RTS (return from subroutine) instruction in it, and then derive the slot number from the return address saved on the stack, as shown in the following example.

```
PHP                ; save status
SEI                ; inhibit interrupts
JSR KNOWNRTS      ; ->a known RTS instruction...
                  ;...that you set up
TSX                ; get high byte of the...
LDA $0100,X       ; ...return address from stack
AND #$0F          ; low-order digit is slot no.
PLP                ; restore status
```

The slot number can now be used in addressing the memory allocated to the peripheral card, as shown in the next section.

I/O addressing

Once your peripheral-card program has the slot number, the card can use the number to address the I/O locations allocated to the slot. Table 2-17 shows how these locations are related to sixteen base addresses starting with \$C080. Notice that the difference between the base address and the desired I/O location has the form \$n0, where *n* is the slot number. Starting with the slot number in the accumulator, the following example computes this difference by four left shifts, then loads it into an index register and uses the base address to specify one of sixteen I/O locations.

```

ASL          ; get n into...
ASL          ;
ASL          ;
ASL          ; ...high-order nybble...
TAX          ; ...of index register.
LDA    $C080,X ; load from first I/O location
    
```

❖ *Selecting your target:* You must make sure that you get an appropriate value into the index register when you address I/O locations this way. For example, starting with 1 in the accumulator, the instructions in the above example perform an LDA from location \$C090, the first I/O location allocated to slot 1. If the value in the accumulator had been 0, the LDA would have accessed location \$C080, thereby setting the soft switch that selects the second bank of RAM at location \$D000 and enables it for reading.

Table 2-17
Peripheral-card I/O base addresses

Base Address	Connector number						
	1	2	3	4	5	6	7
\$C080	\$C090	\$C0A0	\$C0B0	\$C0C0	\$C0D0	\$C0E0	\$C0F0
\$C081	\$C091	\$C0A1	\$C0B1	\$C0C1	\$C0D1	\$C0E1	\$C0F1
\$C082	\$C092	\$C0A2	\$C0B2	\$C0C2	\$C0D2	\$C0E2	\$C0F2
\$C083	\$C093	\$C0A3	\$C0B3	\$C0C3	\$C0D3	\$C0E3	\$C0F3
\$C084	\$C094	\$C0A4	\$C0B4	\$C0C4	\$C0D4	\$C0E4	\$C0F4
\$C085	\$C095	\$C0A5	\$C0B5	\$C0C5	\$C0D5	\$C0E5	\$C0F5
\$C086	\$C096	\$C0A6	\$C0B6	\$C0C6	\$C0D6	\$C0E6	\$C0F6
\$C087	\$C097	\$C0A7	\$C0B7	\$C0C7	\$C0D7	\$C0E7	\$C0F7
\$C088	\$C098	\$C0A8	\$C0B8	\$C0C8	\$C0D8	\$C0E8	\$C0F8
\$C089	\$C099	\$C0A9	\$C0B9	\$C0C9	\$C0D9	\$C0E9	\$C0F9
\$C08A	\$C09A	\$C0AA	\$C0BA	\$C0CA	\$C0DA	\$C0EA	\$C0FA
\$C08B	\$C09B	\$C0AB	\$C0BB	\$C0CB	\$C0DB	\$C0EB	\$C0FB
\$C08C	\$C09C	\$C0AC	\$C0BC	\$C0CC	\$C0DC	\$C0EC	\$C0FC
\$C08D	\$C09D	\$C0AD	\$C0BD	\$C0CD	\$C0DD	\$C0ED	\$C0FD
\$C08E	\$C09E	\$C0AE	\$C0BE	\$C0CE	\$C0DE	\$C0EE	\$C0FE
\$C08F	\$C09F	\$C0AF	\$C0BF	\$C0CF	\$C0DF	\$C0EF	\$C0FF

RAM addressing

A program on a peripheral card can use the eight base addresses shown in Table 2-17 to access the eight RAM locations allocated for its use. The program does this by putting its slot number into the Y index register and using indexed addressing mode with the base addresses. The base addresses can be defined as constants because they are the same no matter which slot the peripheral card occupies.

If you start with the correct slot number in the accumulator (by using the example shown earlier), then the following example uses all eight RAM locations allocated to the slot:

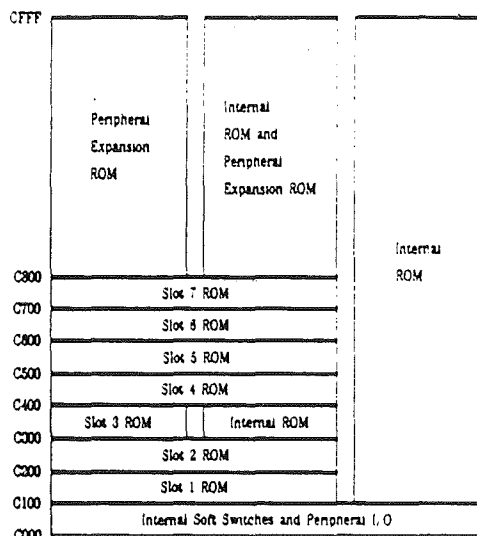
```
TAY
LDA      $0478,Y
STA      $04F8,Y
LDA      $0578,Y
STA      $05F8,Y
LDA      $0678,Y
STA      $06F8,Y
LDA      $0778,Y
STA      $07F8,Y
```

Warning You must be very careful when you have your peripheral-card program store data at the base-address locations themselves since they are temporary storage locations; the RAM at those locations is used by the disk operating system. Always store the first byte of the ROM location of the expansion slot that is currently active (\$Cn) in location \$7F8, and the first byte of the ROM location of the slot holding the controller card for the startup disk drive in location \$5F8.

Other uses of I/O memory space

The portion of memory space from location \$C000 through \$CFFF (decimal 49152 through 53247) is normally allocated to I/O and program memory on the peripheral cards, but there are two other functions that also use this memory space: the built-in self-test firmware and the 80-column display firmware. The soft switches that control the allocation of this memory space are described in the next section.

Figure 2-17
I/O memory map



Switching I/O memory

The built-in firmware uses two soft switches to control the allocation of the I/O memory space from \$C000 to \$CFFF. The locations of these soft switches, SLOTCXROM and SLOTC3ROM, are given in Table 2-18.

◆ *Note:* Like the display switches described in Chapter 2, these soft switches share their locations with the keyboard data and strobe functions. The switches are activated only by writing, and the states can be determined only by reading, as indicated in Table 2-18.

Table 2-18
I/O memory switches

Name	Function	Location			Notes
		Hex	Decimal	Hex	
SLOTC3ROM	Slot ROM at \$C300	\$C00B	49163	-16373	Write
	Internal ROM at \$C300	\$C00A	49162	-16374	Write
	Read SLOTC3ROM switch	\$C017	49175	-16361	Read
SLOTCXROM	Slot ROM at \$Cx00	\$C006	49159	-16377	Write
	Internal ROM at \$Cx00	\$C007	49158	-16378	Write
	Read SLOTCXROM switch	\$C015	49173	-16363	Read

When SLOTC3ROM is on, the 256-byte ROM area at \$C300 is available to a peripheral card in slot 3, which is the slot normally used for a terminal interface. If a card is installed in the auxiliary slot when you turn on the power or reset the Cortland, the SLOTC3ROM switch is turned off. Turning SLOTC3ROM off disables peripheral-card ROM in slot 3 and enables the built-in 80-column firmware, as shown in Figure 6-3. The 80-column firmware is assigned to slot-3 address space

because slot 3 is normally used with a terminal interface, so the built-in firmware will work with programs that use slot 3 this way.

The bus and I/O signals are always available to a peripheral card in slot 3, even when the 80-column hardware and firmware are operating. Thus it is always possible to use this slot for any I/O peripheral that does not have built-in firmware.

When SLOTCXROM is active (high), the I/O memory space from \$C100 to \$C7FF is allocated to the expansion slots, as described previously. Setting SLOTCXROM inactive (low) disables the peripheral-card ROM and selects built-in ROM in all of the I/O memory space except the part from \$C000 to \$C0FF (used for soft switches and data I/O), as shown in Figure 6-3. In addition to the 80-column firmware at \$C300 and \$C800, the built-in ROM includes firmware that performs the self-test of the Cortland's hardware.

- ◆ Note: Setting SLOTCXROM low enables built-in ROM in all of the I/O memory space (except the soft-switch area), including the \$C300 space, which contains the 80-column firmware.

Developing cards for slot 3

Original Ite

In the original Cortland firmware, the internal slot 3 firmware was always switched in if there was an 80-column card (either 1K or 64K) in the auxiliary slot. This means that peripheral cards with their own ROM were effectively switched out of slot 3 when the system was turned on.

With the enhanced Cortland Monitor ROM, the rules are different. A peripheral card in slot 3 is now switched in when the system is started up or when Reset is pressed *if* the card's ROM has the following ID bytes:

\$C305 = \$38

\$C307 = \$18

The enhanced Cortland firmware requires that interrupt code be present in the \$C3 page (either external or internal). A peripheral card in slot 3 must have the following code to support interrupts. After this segment, the code continues execution in the internal ROM at \$C400.

```
$C3F4:IRQDONE STA $C081 ;Read ROM, write RAM
           JMP $FC7A ;Jump to $F8 ROM
           IRQ
           BIT $C015 ;slot or internal ROM
           STA $C007 ;force in internal ROM
```

When programming for cards in slot 3:

- You must support the AUXMOVE and XFER routines at \$C311 and \$C314.
- Don't use unpublished entry points into the internal \$Cn00 firmware, because there is no guarantee that they will stay the same.
- If your peripheral card is a character I/O device, you must follow the Pascal 1.1 firmware protocol, described in the next section.

Pascal 1.1 firmware protocol

The Pascal 1.1 firmware protocol was originally developed to be used with Apple Pascal 1.1 programs. The protocol is followed by all succeeding versions of Apple II Pascal, and can be used by programmers using other languages as well.

The Pascal 1.1 firmware protocol provides Cortland programmers with

- a standard way to uniquely identify new peripheral cards
- a standard way to address the firmware routines in peripheral cards

Device identification

The Pascal 1.1 firmware protocol uses four bytes near the beginning of the peripheral card's firmware to identify the peripheral card.

Address	Value
\$Cs05	\$38 (like the old Apple II Serial Interface Card)
\$Cs07	.\$18 (like the old Apple II Serial Interface Card)
\$Cs0B	\$01 (the generic signature of new cards)
\$Cs0C	\$ci (the device signature)

The first hexadecimal digit, *c*, of the device signature byte identifies the device class; and the second hexadecimal digit, *i*, of the device signature byte is a unique identifier for the card, used by some manufacturers for their cards. Table 2-19 shows the device-class assignments.

Table 2-19
Peripheral-card device-class assignment

Digit	Device class
\$0	Reserved
\$1	Printer
\$2	Joystick or other X-Y input device
\$3	Serial or parallel I/O card
\$4	Modem
\$5	Sound or speech device
\$6	Clock
\$7	Mass storage device
\$8	80-column card
\$9	Network or bus interface
\$A	Special purpose (none of the above)
\$B-F	Reserved for future expansion

For example, the Apple II Super Serial Card has a device signature of \$31: the 3 signifies that it is a serial or parallel I/O card, and the 1 is the low-order digit supplied by Apple Technical Support.

Although version 1.1 of Pascal ignores the device signature, applications programs can use them to identify specific devices.

I/O Routine entry points

Indirect calls to the firmware in a peripheral card are done through a branch table in the card's firmware. The branch table of I/O routine entry points is located near the beginning of the Cs00 address space (s being the slot number where the peripheral card is installed).

The branch table locations that Pascal 1.1 firmware protocol uses are as follows:

Address	Contains
\$Cs0D	Initialization routine offset (required)
\$Cs0E	Read routine offset (required)
\$Cs0F	Write routine offset (required)
\$Cs10	Status routine offset (required)
\$Cs11	\$00 if optional offsets follow; non-zero if not
\$Cs12	Control routine offset (optional)
\$Cs13	Interrupt handling routine offset (optional)

Notice that \$Cs11 contains \$00 only if the control and interrupt handling routines are supported by the firmware. (For example, the SSC does not support these two routines, and so location \$Cs11 contains a nonzero firmware instruction.) Apple II Pascal 1.0 and 1.1 do not support control and interrupt requests, but such requests are implemented in Pascal 1.2 and later versions and in ProDOS.

Table 2-20 gives the entry point addresses and the contents of the 65C02 registers on entry to and on exit from Pascal 1.1 I/O routines.

Table 2-20
I/O routine offsets and registers under Pascal 1.1 protocol

Address	Offset for	X register	Y register	A register
\$Cs0D	Initialization			
	On entry	\$Cs	\$s0	
	On exit	Error code	(unchanged)	(unchanged)
\$Cs0E	Read			
	On entry	\$Cs	\$s0	
	On exit	Error code	(unchanged)	Character read
\$Cs0F	Write			
	On entry	\$Cs	\$s0	Char. to write
	On exit	Error code	(unchanged)	(unchanged)
\$Cs10	Status			
	On entry	\$Cs	\$s0	Request (0 or 1)
	On exit	Error code	(changed)	(unchanged)

Interrupts

The original Cortland offered little firmware support for interrupts. The enhanced Cortland's firmware provides improved interrupt support, very much like the Apple IIc's interrupt support. Neither machine disables interrupts for extended periods.

Interrupts work on enhanced Cortland systems with an installed 80-column text card (either 1K or 64K) or a peripheral card with interrupt-handling ROM in slot 3. Interrupts are easiest to use with ProDOS and Pascal 1.2 because they have interrupt support built in. DOS 3.3 has no built-in interrupt support.

The new interrupt handler operates like the Apple IIc interrupt handler, using the same memory locations and operating protocols. The main purpose of the interrupt handler is to support interrupts in *any* memory configuration. This is done by saving the machine's state at the time of the interrupt, placing the Apple in a standard memory configuration before calling your program's interrupt handler, then restoring the original state when your program's interrupt handler is finished.

What Is an interrupt?

An interrupt is a hardware signal that tells the computer to stop what it is currently doing and devote its attention to a more important task. Print spooling and mouse handling are examples of interrupt use, things that don't take up all the time available to the system, but that should be taken care of promptly to be most useful.

For example, the Cortland mouse can send an interrupt to the computer every time it moves. If you handle that interrupt promptly, the mouse pointer's movement on the screen will be smooth instead of jerky and uneven.

Interrupt priority is handled by a daisy-chain arrangement using two pins, INT IN and INT OUT, on each peripheral-card slot. Each peripheral card breaks the chain when it makes an interrupt request. On peripheral cards that don't use interrupts, these pins should be connected together.

The daisy chain gives priority to the peripheral card in slot 7: if this card opens the connection between INT IN and INT OUT, or if there is no card in this slot, interrupt requests from cards in slots 1 through 6 can't get through. Similarly, slot 6 controls interrupt requests (IRQ) from slots 1 through 5, and so on down the line.

When the IRQ' line on the Cortland's microprocessor is activated (pulled low), the microprocessor transfers control through the vector in locations \$FFFE-\$FFFF. This vector is the address of the Monitor's interrupt handler, which determines whether the request is due to an external IRQ or a BRK instruction and transfers control to the appropriate routine via the vectors stored in memory page 3.

Chapter 3

The FPI Subsystem: New Hardware Features

Introduction

The Fast Processor Interface (FPI) is one of the two major subsystems which make up the Cortland. It is this subsystem that provides these new features for the Apple II family:

- faster processor speed
- support of additional RAM
- I/O shadowing

Figure 3-1 shows the relationship of the FPI within the Cortland. This chapter describes this subsystem and these new functions.

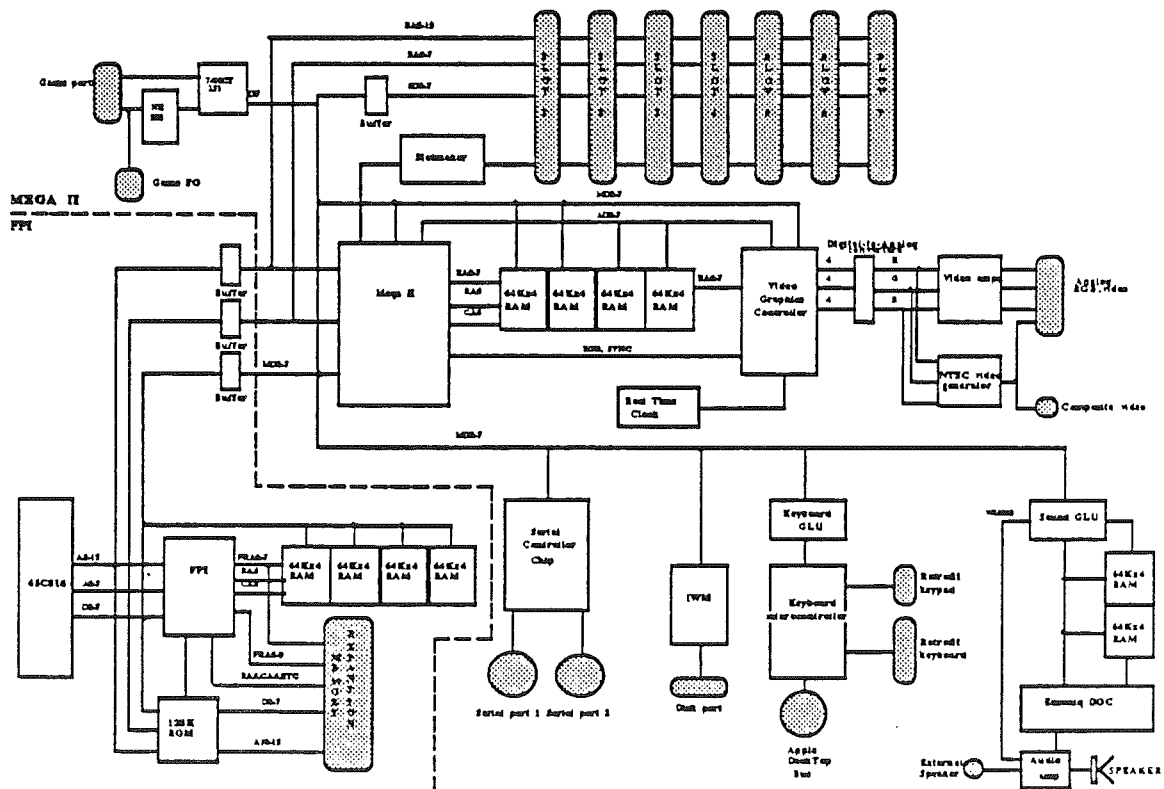


Figure 3-1. The Cortland blockdiagram and the FPI subsystem.

The FPI subsystem

During normal operation, the FPI side of the system runs at 2.8 MHz and the Mega II side runs at 1 MHz. This allows faster processing without disturbing the standard 1 MHz speed for I/O and video timing necessary for compatibility with existing peripherals.

Data must be transferred and soft switches accessed between the Mega II and the FPI. When the CPU must access an I/O or Mega II RAM location, the system slows down briefly to 1 MHz and synchronizes itself with the Mega II timing so that the access can be accomplished. When the access is finished, the FPI side returns to the normal 2.8 MHz operating speed, and the Mega II continues at 1 MHz.

A combination of existing Apple II soft switches, Mega II soft switches, and FPI control registers control the various functions of the FPI. The control registers include

- the State register
- the Shadow register
- the Configuration register

Standard Apple II Soft Switches

To ensure compatibility with existing Apple II software, the Cortland includes the standard Apple II soft switches. These soft switches are available in any 64K bank where shadowing is enabled AND when the I/O enable bit in the Shadow register is set to zero.

While most of the soft switches reside in the Mega II, some of the soft switches are duplicated in the FPI to provide ROM control, language-card operation, and video shadowing. The soft switches in the FPI are write-only. The soft switches duplicated in the FPI are

- 80STORE
- RAMRD
- RAMWRT
- SLOTC3ROM
- INTCXROM
- ALTZP
- ROMBANK
- PAGE2
- HIRES
- SLOT-3 device-select switch

For detailed descriptions of each of the standard Apple II soft switches, see *Chapter 2: The Apple II Environment*.

Memory Allocation

The FPI controller can access a minimum of 128K bytes of RAM and is expandable to 8MB. This RAM is separate from the 128K RAM available to the Mega II, and is used for text and graphics display buffers, and system software. The FPI also has access to 128K bytes of ROM, expandable to 1MB. The application program is free to use the remaining locations in banks \$0 and \$1, and those in the RAM expansion banks. Figure 3-2 shows the system memory map.

MEMORY MAP FIGURE

Figure 3-2. Memory map.

The State Register

The State register is duplicated with in FPI IC, allowing access to eight of the commonly-used soft switches in a single transfer as shown in Figure 3-3. This means that reading the State register at location \$C086 will allow you to read the state of the soft switches without slowing the system. Write operations to the State register will slow the system momentarily. Figure 3-3 shows the soft switches in the State register.

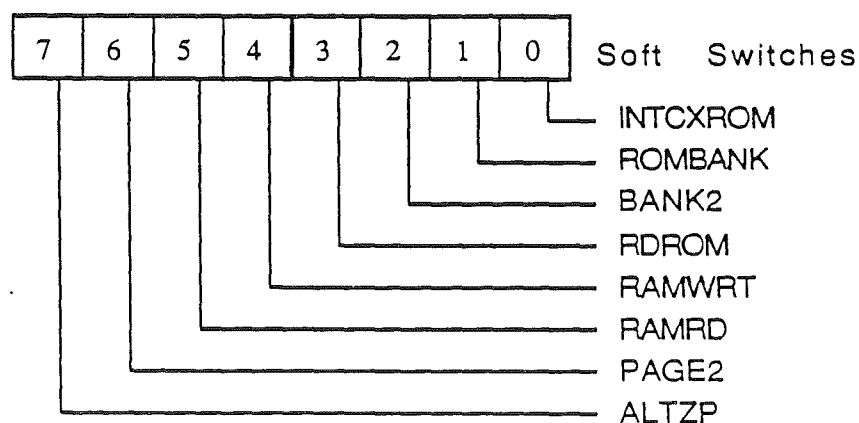


Figure 3-3. State register at \$C068.

Shadowing

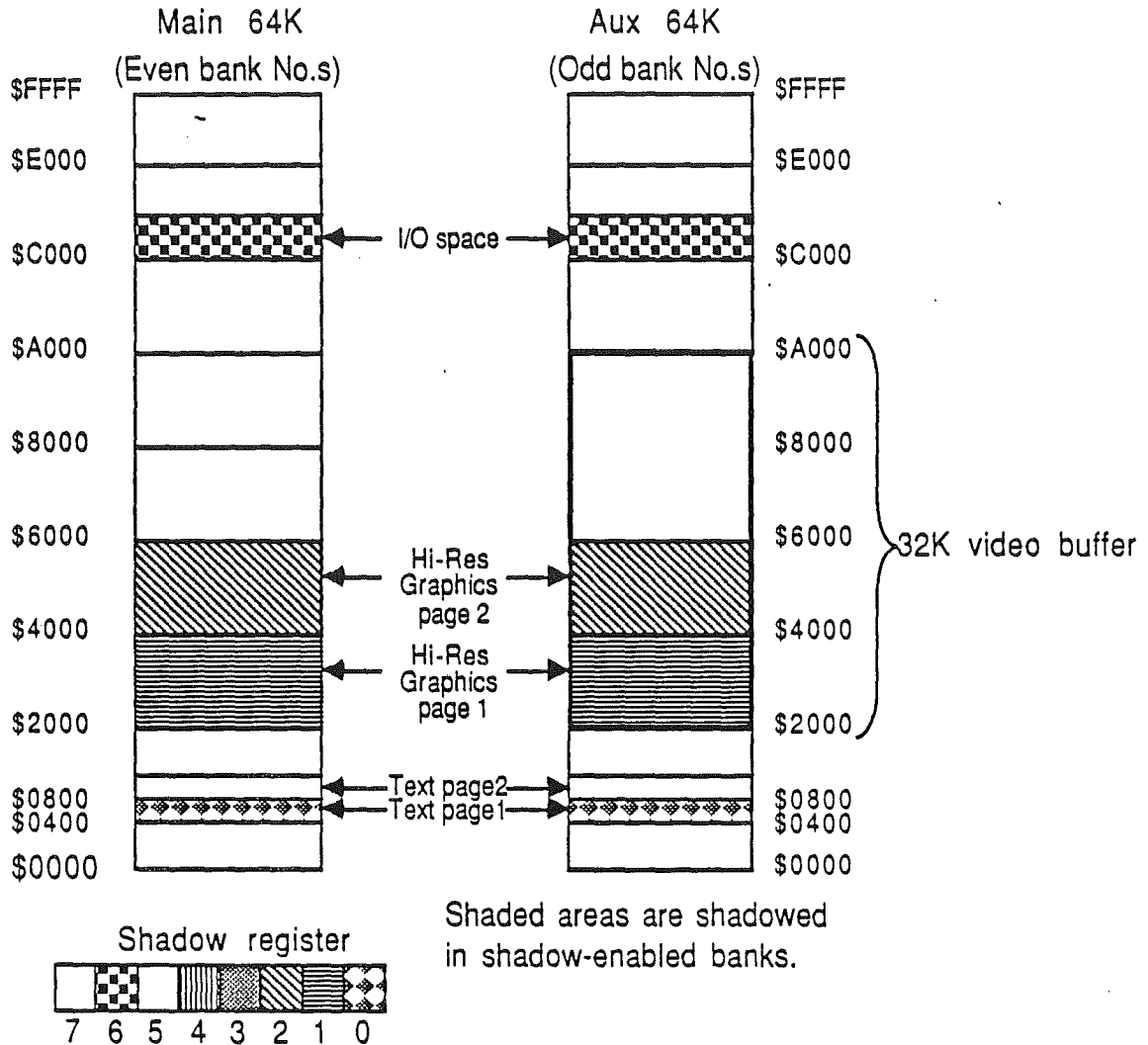
Shadowing is the process of duplicating the I/O (input/output) addresses in another bank of RAM. Shadowing may be enabled in any RAM memory bank. The original I/O addresses are located in banks \$E0 and \$E1, which is low-speed Mega II main address space. By shadowing these locations in the high-speed FPI address space, only writes to the I/O location require a reduction in system speed.

An I/O write actually writes to the I/O address in both banks, the Mega II bank (\$E0 or \$E1, depending upon the type of access) and the FPI bank (\$00 through \$7F) that has shadowing enabled. Shadowing, therefore, helps minimize the impact of video display updates on the overall system speed: only I/O writes are done at low system speed; I/O reads are done at full system speed. Here are the shadowing options:

- Enable shadowing for banks \$00 and \$01 only
- Enable shadowing for all RAM banks (\$00 thru \$7F)

Enable shadowing by setting bit 4 in the Configuration register (described in detail on page 18) to 1. This will duplicated the I/O locations in all high-speed RAM banks. Reads and writes can now be done from and to these shadowed locations. Direct access to banks \$E0 and \$E1 is not inhibited by shadowing; direct access to the video buffers and I/O is still available through these Mega II banks.

Memory banks \$00 and \$01 contain the I/O space. Portions of these banks are duplicated into higher-numbered memory banks when shadowing is enabled. Figure 3-4 shows banks \$0 and \$1, or any even-odd pair of banks, if shadowing is selected in the Configuration register.



The shadow register above shows which bits control shadowing of these areas in banks where shadowing is enabled.

Figure 3-4. Shadowed Memory Map

The Shadow Register

The Shadow register controls which address ranges of each shadowed high-speed RAM bank are duplicated in the Mega II RAM display areas. The Shadow register also determines whether or not the I/O space/language-card areas for each bank is implemented. Figure 3-5 shows the format of the Shadow register, followed by a list of each of the bits and their function.

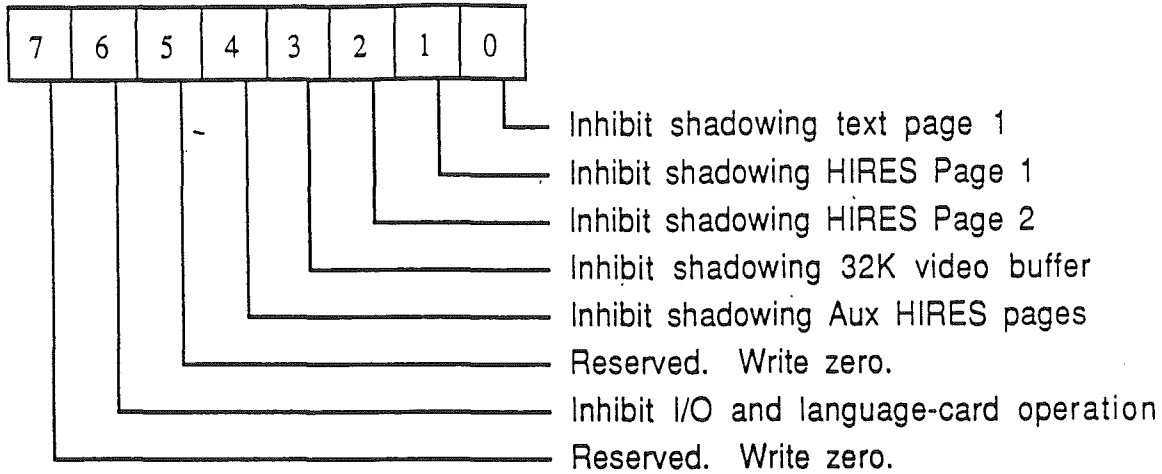


Figure 3-5. Shadow Register at \$C035.

Bit	Value	Description
7	-	Not used. Write 0.
6	0	The I/O and language-card (IOLC) inhibit bit. This bit controls whether the 4K range from \$C000 to \$CFFF acts as RAM or I/O. When this bit is 0, I/O is enabled in the \$Cxxx space and the RAM that would normally occupy that space becomes a second \$Dxxx RAM space, forming a language card.
	1	When this bit is 1, the I/O space and language card are inhibited, and contiguous RAM is available from \$0000 through FFFF.

For more information on I/O and the language card, see Chapter 2: Apple II Compatibility.

5	-	Not used. Write 0.
4	1	Inhibit shadowing for Auxiliary Hi-Res pages. When this bit is 1, all shadowing enabled for Hi-Res video pages 1 and 2 (as determined by bits 0 through 3 in this register) is disabled for all shadowed Aux (odd) banks. Shadowing of main bank Hi-Res video pages remains unaffected.
	0	When this bit is 0, all shadowing enabled for Hi-Res video pages (as determined by bits 0 through 3 in this register) is enabled for aux bank Hi-Res video pages as well.
3	1	Video buffer inhibit. When this bit is 1, shadowing is disabled for the entire 32K video buffer.
	0	When this bit is 0, shadowing is enabled for the 32K video buffer.
1	1	Hi-Res video page 2 inhibit. When this bit is 1, shadowing is disabled for Hi-Res video page 2 and aux Hi-Res video page 2.
	0	When this bit is 0, shadowing is enabled for Hi-Res video page 2 and aux Hi-Res video page 2, unless auxiliary page Hi-Res shadowing is prohibited by bit 4 of this register.

- 1 1 Hi-Res video page 1 inhibit. When this bit is 1, shadowing is disabled for Hi-Res video page 1 and aux Hi-Res video page 1.
- 0 0 When this bit is 0, shadowing is enabled for Hi-Res video page 1 and aux Hi-Res video page 1, unless auxiliary page Hi-Res shadowing is prohibited by bit 4 of this register.
- 0 1 Text page 1 inhibit. When this bit is 1, shadowing is disabled for text page 1 and aux text page 1.
- 0 0 When this bit is 0, shadowing is enabled for text page 1 and aux text page 1.

Note: Text page 2 (\$0800 through 0BFF) is never shadowed. If you need a text display area or a code storage area, use Mega II banks \$E0 and E1. These banks are limited to 1 MHz operation, however.

Areas within each shadow-enabled 64K bank may be shadowed or not by setting the corresponding bit or bits in the Shadow register. Shadowing may be turned off (no banks shadowed) by setting all bits in the Shadow register. When the Shadow register is cleared on reset, it defaults to shadowing all video areas.

Each bit in the Shadow register is active high, which means that the shadowing of the selected area is inhibited if the corresponding bit is set. Programs that use the Shadow register can turn off shadowing in unused video areas by setting the appropriate bits and reclaim the free memory space in the unused video buffers in Mega II banks \$E0 and E1.

The Configuration Register

The Configuration register contains bits that control the speed of operation and that determine whether specific area within a bank is shadowed. The Configuration register is cleared on reset or power up. Figure 3-6 shows the format of the Configuration register, followed by a description of each bit.

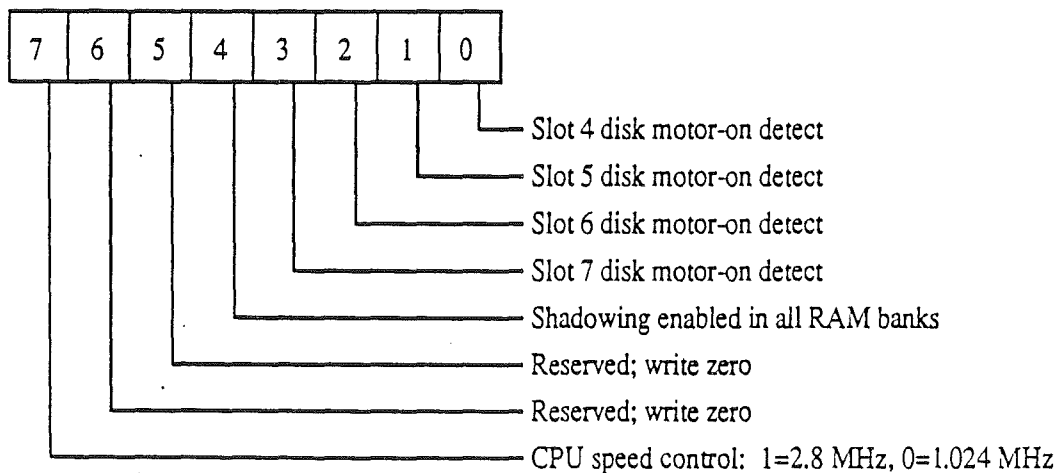


Figure 3-6. Configuration Register

Bit Value Description

7	1	System operating speed. When this bit is 1, the system operates at 2.8 MHz.
	0	When this bit is 0, the system operates at 1.024 MHz (as in an Apple II).
5-6	-	Not used; set to 0.
4	1	Bank shadowing bit: this bit determines memory shadowing in the RAM banks. Bits 0 through 3 will determine which portions, if any, of the banks will be shadowed. To enable shadowing in all RAM banks \$00 through 7F, set this bit to 1.
	0	To enable shadowing in banks \$00 and \$01 only, clear this bit.
0-3	1	Disk II motor address detectors: to retain Apple II peripheral-compatibility, the motor-on detectors slow the system to 1.024 MHz whenever the disk motor-on address is detected. When the disk motor-off address is accessed, the system speed increases up to 2.8 MHz again. For example, when bit 1 is 1, the FPI switches to slow mode (1.024 MHz) when address \$C0D9 is accessed, and returns to normal speed (2.8 MHz) following a \$C0D8 access. (See list of addresses below.)
	0	When this bit is 0, the disk II motor detector is turned off.

Bits 0 through 3 detect the following address:

Slot	Motor-on	Motor-off
4	\$C0C9	\$C0C8
5	\$C0D9	\$C0D8
6	\$C0E8	\$C0F9
7	\$C0F8	\$C0F9

Note: Drives designed for the Cortland system should use the speed bit (Configuration register bit 7) to change the processor speed when accessing disks, rather than the disk motor-on detectors (Configuration bits 0 through 2). By using bit 7, you access drives in slots other than slots 4 through 7 by changing the system speed manually. Be aware that CPU speed changes for drive compatibility may affect application program timing; avoid using the motor addresses unless they are used in a fashion consistent with the drive's CPU speed requirements.

RAM Control

The FPI alone controls the high-speed RAM. This high-speed memory consists of a minimum of 128K RAM on the motherboard and additional expansion RAM on the extended memory card for a total of 8MB.

The FPI provides memory refresh for the high-speed RAM which incorporate internal refresh-address counters. This refresh scheme frees the address bus so that the FPI can execute ROM cycles while RAM refresh cycles are occurring thus, allowing full speed operation in the ROM. These cycles occur approximately every 3.5 μ s and reduce the 2.8 Mhz processing speed by approximately 8% for programs that run in RAM. When running at 1.024 MHz, refresh cycles are executed during an unused portion of the processor cycle and do not effect the processor speed.

ROM

The FPI provides control for 128K of on-board ROM and additional expansion-card ROM for a total of 1MB. The Cortland on-board system ROM is located in banks \$FE through \$FF. Banks \$F0 through \$FD are reserved for ROM expansion. ROM that occupies this address space may reside only on the extended memory card, along with additional expansion RAM.

I/O Processing

Normally, all I/O write accesses are in the designated I/O space and are written to the Mega II I/O space as well as that in the FPI. However, when FPI internal registers (the DMA bank register, the Configuration register, and the Shadow register) are accessed, or when the interrupt ROM addresses (\$C071 through 7F) are read, only the FPI is written to.

Two of the FPI registers that are duplicated in the Mega II are the State register and the Slot ROM register. Any writes to these registers are also written to the Mega II (and the system speed is slowed momentarily). Any reads access the FPI only.

The interrupt ROM code is available when shadowing is enabled and the IOLC bit in the Shadow register is set. The INTCXROM soft switch does not effect interrupt ROM accesses.

The Slot Register

The Slot register is used to select which device is enabled for each of the seven logical slots. That device can be either the internal or the peripheral slot device. If the enable bit is one, accesses for that slot ROM space (\$Cnxx) are directed to the ROM on the slot card. If the enable bit is cleared, the built-in I/O device is selected, and the system ROM code associated with the slot is executed.

The control panel (accessible by pressing the Apple-Control-Reset keys simultaneously) also allows the user to select the appropriate device for each logical slot. The bit representations are shown in figure 3-7, followed by list of the bit descriptions.

Note: Slot 4 device hardware addresses are always available. However, the slot 4 ROM space is controlled by the SLOTC3ROM switch to maintain compatibility with the existing Apple II products.

Note: I/O space for slots 3 (\$C0C0 to \$C0CF) and 4 (\$C0D0 to \$C0DF) is always enabled.

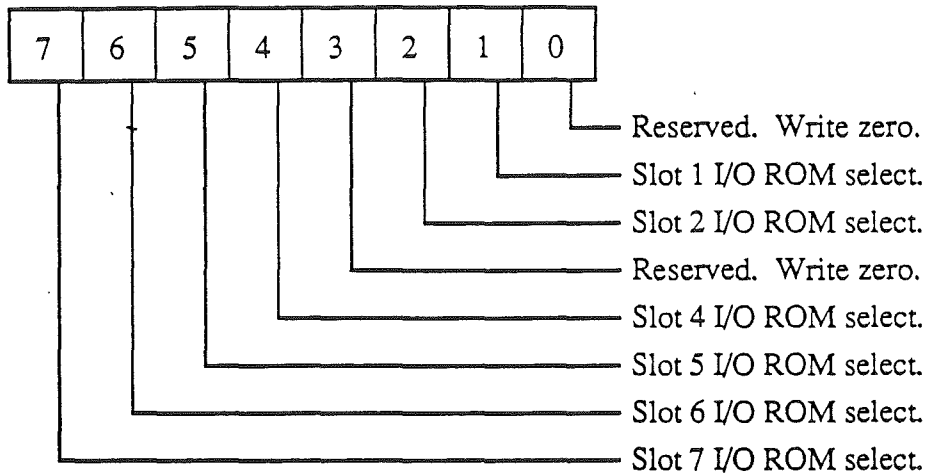


Figure 3-7. The Slot register at \$C02D.

Bit	Value	Description
7	0	Selects the internal-device (AppleTalk) ROM code for slot 7 is selected.
	1	Enables both the slot-card ROM space (location \$C700 to \$C7FF) and I/O space \$C0F0 to \$C0FF.
6	0	Selects the internal-device (5-1/4" disk drive) ROM code for slot 6.
	1	Enables both the slot-card ROM space (location \$C600 to \$C6FF) and I/O space \$C0E0 to \$C0EF.
5	0	Selects the internal-device (3-1/2" disk drive) ROM code for slot 5.
	1	Enables both the slot-card ROM space (location \$C500 to \$C5FF) and I/O space \$C0D0 to \$C0DF.
4	0	Selects the internal-device (mouse) ROM code for slot 4.
	1	Enables both the slot-card ROM space (location \$C400 to \$C4FF).
3	-	Reserved. Write zero. Slot 3 internal device (80-column circuitry) ROM code is always selected.
2	0	Selects the internal-device (serial port) ROM code for slot 2.
	1	Enables both the slot-card ROM space (location \$C200 to \$C2FF) and I/O space \$C0A0 to \$C0AF.
1	0	Selects the internal-device (serial port) ROM code for slot 1.
	1	Enables both the slot-card ROM space (location \$C100 to \$C1FF) and I/O space \$C090 to \$C09F.
0	-	Reserved. Write zero.

Synchronization

Whenever data needs to be transferred between the FPI and the Mega II, the FPI IC must first fall into step, or synchronize with the slower-running Mega II. This may be a single, Mega II cycle as when a single I/O location in the Mega II must be accessed, or consecutive Mega II cycles (extended periods of low-speed operation) as when Apple II software must be run at the lower speed for compatibility. The FPI runs the CPU at low-speed by generating one CPU cycle for each Mega II cycle, thus running the CPU at precisely 1.024 MHz. This is necessary to support time-dependent Apple II software.

In all Apple II products, every 65th CPU cycle is elongated, or stretched by 140ns. This is required to keep the video display consistent.

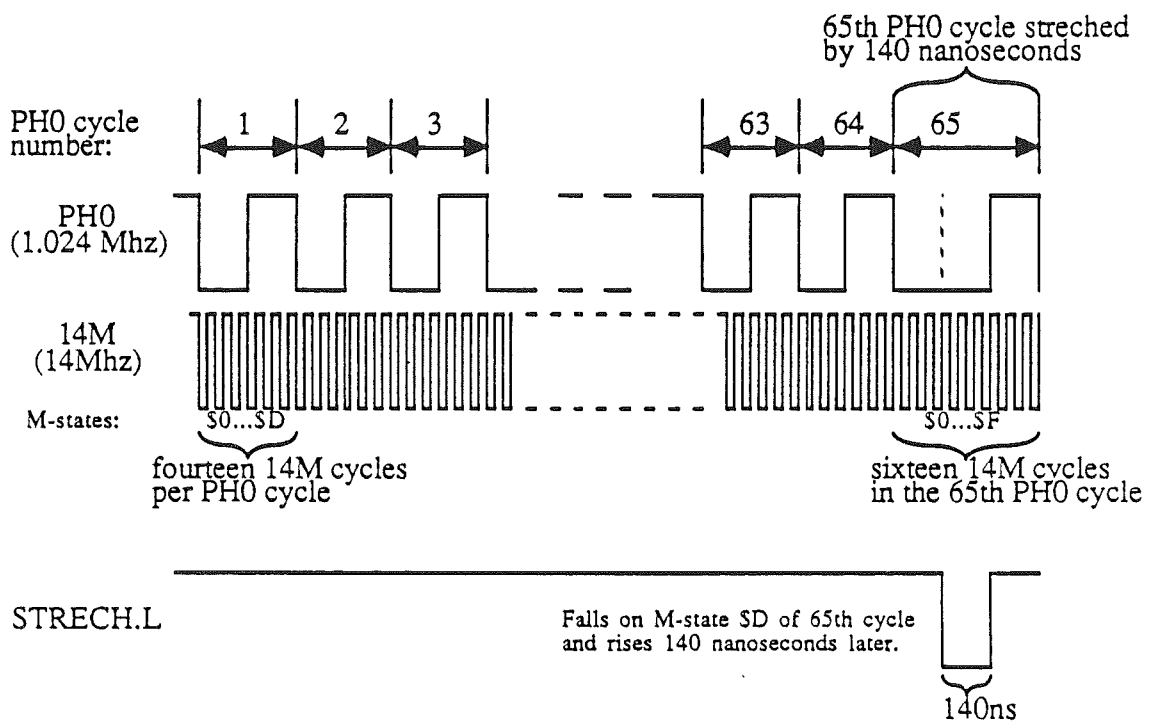


Figure 2-8. PH0 cycles, 14M cycles and M-States.

To help keep track of where the Mega II is in its cycle, the FPI uses an internal counter called the M-State counter, which counts the 14M cycles within one PH0 cycle. There are 14 (SE) M-States (14M cycles) per PH0 cycle. Figure 2-8 shows the relationship between the PH0 cycles and the M-States.

The FPI takes advantage of the Apple II standard of stretching every sixty-fifth clock cycle in order to simplify the synchronization procedure. Synchronization is achieved first by slowing down the CPU. Next, when the M-State counter reaches a count of SD during the sixty-fifth Mega II cycle, the VGC pulls STRETCH.L signal (an input to the FPI) low. Also, when the system is first powered on, STRETCH.L is asserted. This forces the M-State counter to begin at a SD count.

Note that STRETCH.L is asserted during the sixty-fifth PH0 cycle, and coincides with the stretched CPU cycle that is used in all Apple II products. If the FPI cycle, although slowed to 1.024 MHz, were not also stretched to coincide with the stretched Mega II cycle, the two devices would not be synchronized for more than 980 μ s (65 PH0 cycles).

Mega II Cycle

A Mega II cycle is a CPU or DMA cycle that requires access to the low-speed side of the system. These are

- all external and most internal I/O operations
- shadowed video write operations
- *inhibited* memory accesses
- Mega II memory accesses to banks \$E0 and E1.

Direct Memory Access is a means of providing fast I/O. A peripheral card in one of the expansion slots can require DMA. For more information on DMA, see *Chapter 7: Expansion Slots*.

A Mega II cycle consists of these steps:

1. A Mega II cycle begins when the FPI recognizes an address that requires access to the slow side of the system, such as listed above.
2. Approximately 90 nanoseconds after the CPU PH2 clock goes low, the location address and bank address from the CPU becomes valid. The FPI decodes these addresses and determines the type of cycle to be executed before the PH2 clock rises.
3. If the cycle is a Mega II cycle, the FPI holds the PH2 clock high, and remains in this state until it reaches the end of M-State 1.
4. On the next rising clock edge, the FPI sets the internal SYNC flag to indicate that it is now synchronized with the Mega II timing.
5. The memory or I/O access takes place.

Mega II Aux Bank Access

To allow direct access to the Mega II auxiliary bank, the FPI passes the least significant bit (lsb) of the bank address to the Mega II during each Mega II cycle. If shadowing is enabled or the software is addressing bank \$E0 or E1, an odd-numbered bank address will access the Mega II auxiliary memory automatically, without using the soft switches. For this setup to work, the programmer must first set bit 0 to 1 in the video-control register at \$C029. Otherwise, the Mega II will ignore the bank bit, and the soft switches must then be used to access the auxiliary 64K through an even-numbered, shadowed bank.

Real Time Clock IC Interface

In addition to the many video tasks the VGC performs, it also functions as an interface between the microprocessor and the Real Time Clock (RTC) chip. This chip provides the system with calendar and clock information as well as parameter RAM preserved by battery power. These functions are performed through two read/write registers, the control and data registers.

The Control register (located at \$C034), shown in figure 4-11, serves a dual function: as the command register for the RTC and as the border color register. Refer to *Screen Border Color* in this chapter for more information on controlling the color of the display border.

Serial data communication between the VGC and the RTC are carried out one byte at a time. (The terms "read" and "write" are used in perspective of the VGC: a read transfers data from the clock chip to the VGC while a write transfers data from the VGC to the clock chip.) To write to the clock chip, the program must first write the data into the data register (\$C033), then set the appropriate bits in the control register (\$C034). To read from the clock chip, set the appropriate control register bits, and then read the data from the data register.

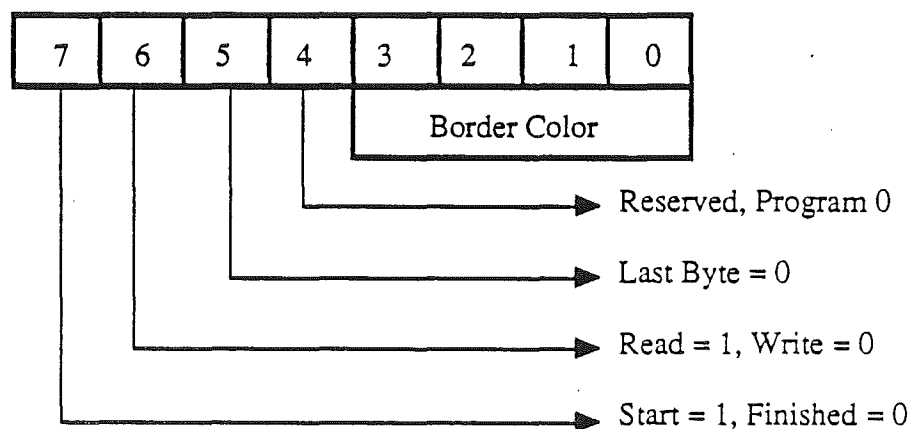


Figure 4-11. Control register at \$C034.

Bit	Value	Description
7	1	A read or write to the the Clock chip begins by setting this bit to 1.
	0	This bit is set to 0 automatically by the RTC when the data exchange is complete. The program can detect that the exchange has been completed by polling bit 7 for a 0.
6	1	The read/write bit. Set this bit to 1 prior to a read from the RTC
	0	Set this bit to 0 prior to a write to the RTC.
5	1	After the last byte has been read or written, this bit must be set to 1. This last step is necessary to avoid corrupting the data in the clock chip after the transactions are completed.
	0	The last-byte control bit. A data transfer typically involves an exchange of two or three bytes. Set this bit to 0 before transferring any bytes to or from the RTC.

Cortland Hardware Reference Manual

- 4 - Not used; set to 0.
- 3-0 - Display border color. See *Screen Border Color* in this chapter for details on selecting the video display border color.

Chapter 4

Apple Cortland Video

Introduction

The Apple Cortland can display several video modes. These include display modes that are compatible with the rest of the Apple II family (as well as some enhancements to these existing modes), and some completely new modes. These new video modes provide higher resolution, greater color flexibility, and programming ease previously unseen in the Apple II product line. Figure 4-1 shows a block diagram of the Apple Cortland and the video components. This chapter describes the enhanced Apple II-compatible video, the new video enhancements, and the new video display modes.

Apple II compatibility

The Apple Cortland shares several display modes with previous Apple II computers. The Cortland supports and enhances these existing Apple II video modes:

- 40-column and 80-column text modes
- Mixed text/graphics mode
- Low-resolution graphics mode
- High-resolution graphics mode
- Double high-resolution graphics mode

For more information on Apple II-compatible video, refer to *Chapter 2: Apple II Compatibility*.

The Apple Cortland adds these enhancements to the existing Apple II-video modes:

- the ability to select unique text and background colors from any of the following 16 Apple II colors:

\$0	black
\$1	magenta
\$2	dark blue
\$3	purple
\$4	dark green
\$5	dark gray
\$6	medium blue
\$7	light blue
\$8	brown
\$9	orange
\$A	light gray
\$B	pink
\$C	green
\$D	yellow
\$E	aquamarine
\$F	white

Table 4-1. Text and background colors.

- the ability to select the border color for the perimeter of the video image. You can choose this color from any of the 16 Apple II colors listed in table 4-1.
- the ability to display gray-scale video: this means that you can display color video output on monochrome monitors in shades of gray rather than in dot patterns that represent color. This ability increases contrast between graphics colors on a monochrome monitor.

Removing color from the composite video signal in 40-column and 80-column text modes makes text more readable. Color is not removed when the computer is running in mixed text/graphics modes, and the four lines of text at the bottom of the display will exhibit **color fringing on composite color monitors.**

Color fringing is the rainbow-like effect that appears around text characters when they are displayed in color on most color monitors. This fringing is unavoidable because the color detection circuitry of most composite color monitors cannot respond fast enough to the changing of the color information during the text portion of the display. Displaying text in black-and-white makes it more readable.

The Video Graphics Controller

The Video Graphics Controller (VGC) custom IC is responsible for generating all video displayed by the Apple Cortland. The VGC provides the following functions:

- supports and enhances existing Apple II video modes
- implements new video modes
- provides interrupt handling for three interrupt sources

The VGC generates all video output in all video modes. The Mega II custom IC is responsible for maintaining the video RAM; all writes to the video display buffer are done

via the Mega II. Figure 4-1 shows the relationships of the VGC, Mega II, main, and auxiliary RAM.

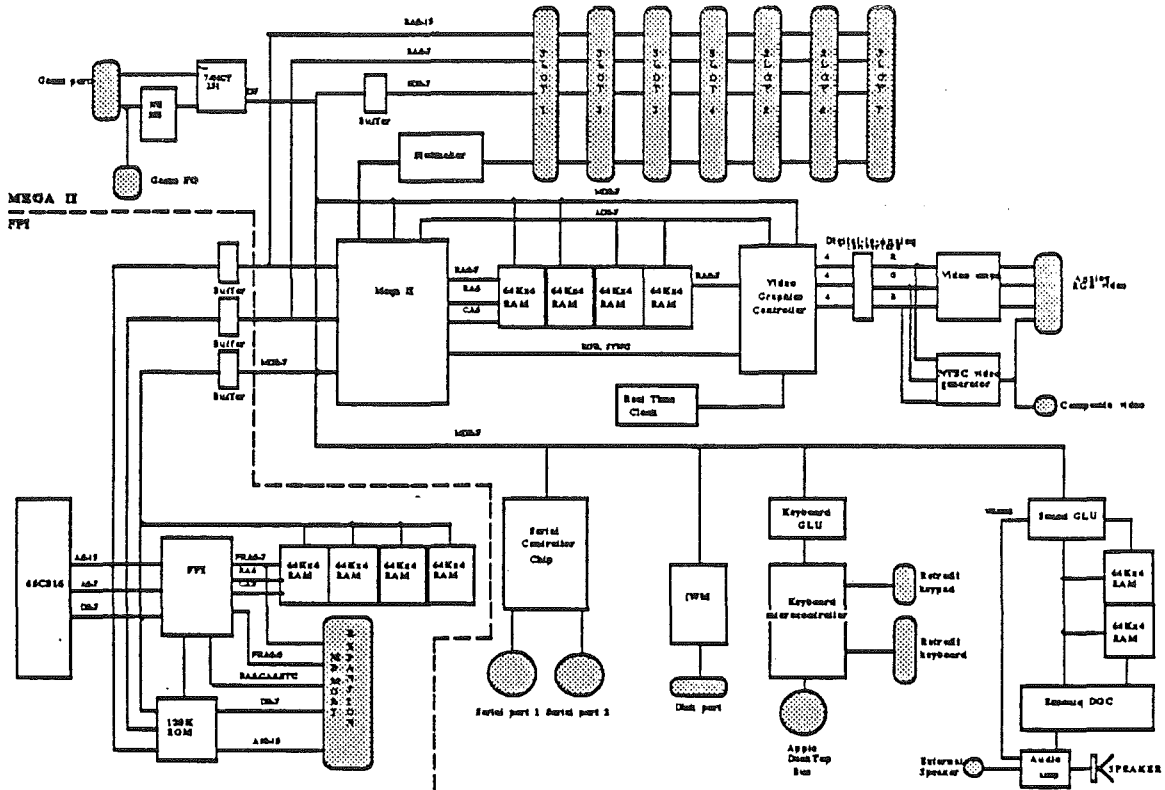


Figure 4-1. Block diagram of the Apple Cortland and video components.

New video display enhancements

The Apple Cortland provides new enhancements to the existing Apple II video modes. These include

- selectable screen-border color
- selectable background color
- selectable text color
- selectable color or black-and-white video

These enhancements are described below. The new graphics modes— Super Hi-Res graphics and color-fill graphics— are described in the next section.

Text and Background Color

The Apple Cortland provides the capability of colored text on a colored background. To select one of these new display options, write the appropriate color values to the Screen Color register located at \$C022.

The Screen Color register is an 8-bit dual-function register. First, the most significant four bits determine the text color. Second, the least significant four bits determine the background color. You can choose these colors from the sixteen available Apple II colors given in table 4-1. The user can also select these colors from the Control Panel. Figure 4-2 shows the format of the Screen Color register, followed by a description of each bit in the register.

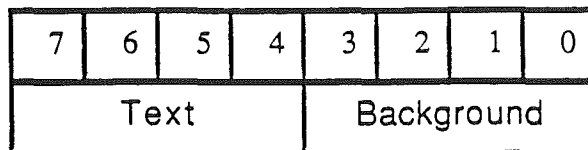


Figure 4-2. Screen Color register at \$C022.

Bit value description

- 7-4 - Text color
- 3-0 - Background color

Border color

The colored border area surrounds the video display text area. You may select a color for the border by writing the appropriate color value to the Screen Border register located at \$C034. You can choose this color from the sixteen Apple II colors listed in Table 4-1. Alternately, the user can select the border color from the control panel.

The Border Color register is an 8-bit read/write register serving two functions. First, the least significant four bits determine the border color. Second, the most significant three bits are the control bits for the Real Time Clock chip (RTC) interface logic. Figure 4-3 shows the Border Color register format. A description of each bit follows the figure.

Note: When you change the border color, it is important to use values between 00 and \$0F when writing to this register. This will insure the RTC chip contents remains uncorrupted.

See the section on the Real Time Clock Interface in Chapter 3: *New Cortland Features* for more information on the RTC.

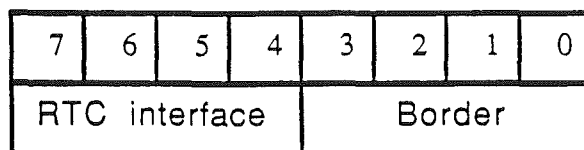


Figure 4-3. Border Color register at \$C034.

Bit Value Description

- 7-4 - Real Time Clock control bits (see *Real Time Clock* section).
- 3-0 - Border color.

To color or not to color...

The Apple Cortland video is displayed in either color or black-and-white. Figure 4-4 shows the format of the Monochrome/Color register, followed by a description of each bit. Located at \$C021, this register controls whether the composite video signal consists of color or gradations of gray. If bit seven is a 1, video displays in black-and-white; if it is a 0, video displays in color.

If using a monochrome monitor, set this bit to 1. Displaying text in black-and-white results in a better-looking, more readable display. This bit does not affect the RGB outputs. The remaining bits are not used; set them to 0 when writing to this location. The user can also select color or monochrome video from the control panel. Figure 4-4 shows the format of the Monochrome/Color register, followed by a description of each bit in the register.

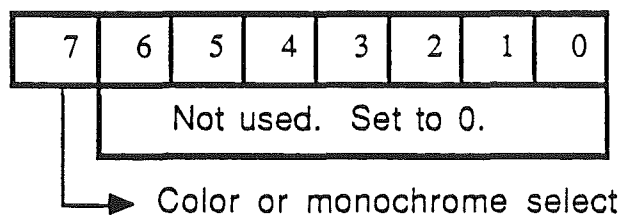


Figure 4-4. Monochrome/Color register at \$C021.

Bit	Value	Description
7	1	Composite gray scale video output.
	0	Composite color video output.
6-0	-	Not used. Set to 0.

New graphics display modes

The Cortland has two video modes that are new to the Apple II family. These are the 320-pixel and 640-pixel Super Hi-Res graphics modes which increase horizontal resolution to either 320 or 640 pixels and increase vertical resolution to 200 lines.

A pixel is the smallest individually addressable video, or picture element (hence the word pixel). The Apple Cortland video screen displays different quantities of pixels, depending upon the video mode. In Super Hi-Res graphics 640-mode, the screen contains 128,000 pixels (640 pixels on each of 200 lines). In 320-mode, half that number of pixels are displayed.

Another new feature of Cortland video graphics is Color-Fill, an option which simplifies the task of painting continuous color on any one line.

Color-Fill mode lets you draw consecutive pixels on a scan line in the same color faster and much more conveniently than possible previously.

Super Hi-Res graphics

The Cortland uses Super Hi-Res graphics to implement new video graphics features, previously unavailable in the Apple II family of computers. The VGC is primarily responsible for supporting the Super Hi-Res video graphics, which provides these new video capabilities:

- 320- or 640-horizontal resolution selectable
- 200-line vertical resolution
- 12-bit color resolution that allows 4096 available colors to choose from
- 16 colors for each of the 200 lines —up to 256 colors per frame
- Color-Fill mode
- Scan-Line interrupts
- all new video mode features programmable for each scan line
- linear display buffer
- pixels contained within byte boundaries

The Super Hi-Res graphics buffer

The Super Hi-Res Graphics display buffer contains three types of data: pixel data, pointer data, and color palettes. Figure 4-5 shows a memory map of the display buffer. This buffer resides in contiguous bytes of the auxiliary 64K bank of the slow RAM from \$2000 through \$9FFF. Note that this display buffer uses memory space used for the Apple II Double Hi-Res graphics buffers, but leaves the other graphics and text display buffers untouched.

The next three paragraphs describe the color palette, pointer, and pixel data bytes used in Super Hi-Res graphics mode.

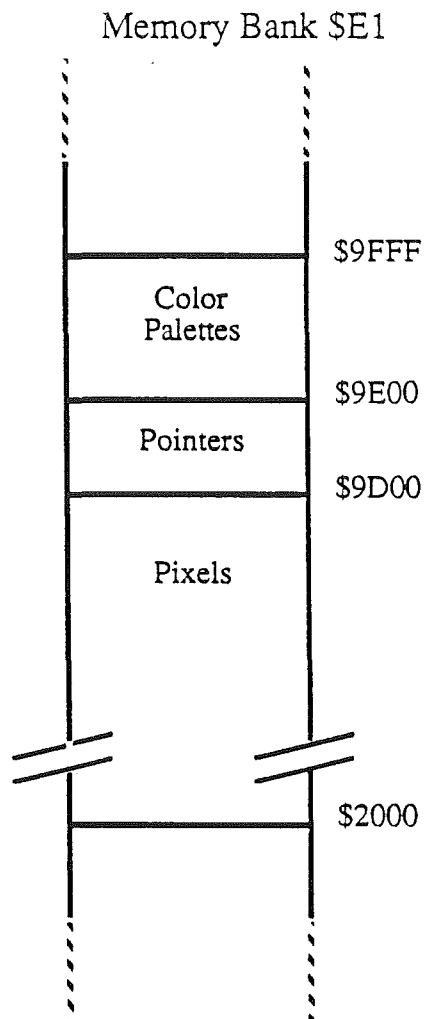


Figure 4-5. Super Hi-Res graphics display buffer

Pointer Bytes (\$9D00-\$9DFF)

An added advantage of the new Apple Cortland video graphics is the ability to enable or disable the Super Hi-Res graphics feature for each video scan line. The pointer bytes (located from \$9D00 through \$9DFF as shown in Figure 4-6) control the features for each scan line. There are 200 pointer bytes, one 8-bit byte for each of the 200 scan lines. For each line, you can select

- the palette (16 colors) to be used on the scan line
- Color Fill mode on the scan line
- an interrupt to be generated on the scan line
- either 320-pixel or 640-pixel resolution for the scan line

The pointer-byte bits and their functions are listed in Figure 4-6, and a description of each follows.

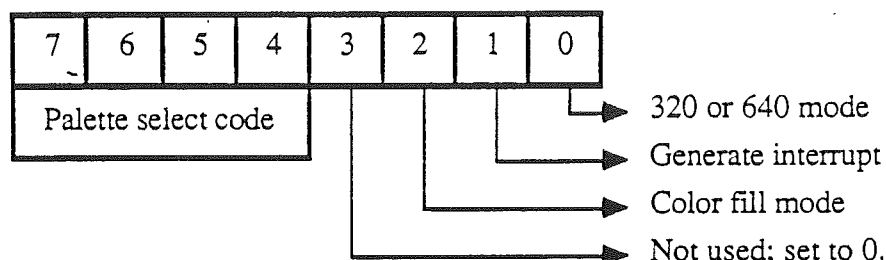


Figure 4-6. Pointer Byte at \$9DXX (see text below).

Bit	Value	Description
7-4	-	Palette chosen for this scan line.
3	-	Not used; must be set to 0.
2	1	color-fill mode enabled. This mode is available only in Super Hi-Res 320-pixel resolution mode. In 640-pixel mode, Color-Fill mode is disabled.
	0	color-fill mode disabled.
1	1	Interrupt generated for this scan line. When this bit is a 1, the Scan-Line interrupt status bit is set at the beginning of the scan line.
	0	Scan line interrupts disabled for this scan line.
0	1	Horizontal resolution = 640 pixels.
	0	Horizontal resolution = 320 pixels.

The location of the pointer byte for each scan line is \$9DXX, where XX is the hexadecimal value of the line. For example, the pointer byte for the first scan line (line 0) is located in memory location \$9D00; the pointer byte for the second scan line (line 1) is in location \$9D01, and so forth.

Note: The first 200 bytes of the 256 bytes in the memory page beginning at \$9D00 are pointers bytes, and the remaining 56 bytes are reserved for future expansion. For compatibility with future Apple II products, always set these 56 bytes to zero.

Color Palettes (9E00-\$9FFF)

A color palette is a group of 16 colors to be displayed on the scan line. Each scan line can have a different palette assigned to it. These 16 colors can be chosen from any of the 4096 colors available. You can draw each pixel on the scan line in any of the 16 colors that make up the palette.

These colors are determined by a 12-bit value made up of three separate 4-bit values. Each 4-bit quantity represents the intensity of each red, green and blue. The combination of the magnitudes of each of the three primary colors determines the resulting color. Figure 4-7 shows the format of each of these 4-bit values that make up a palette color.

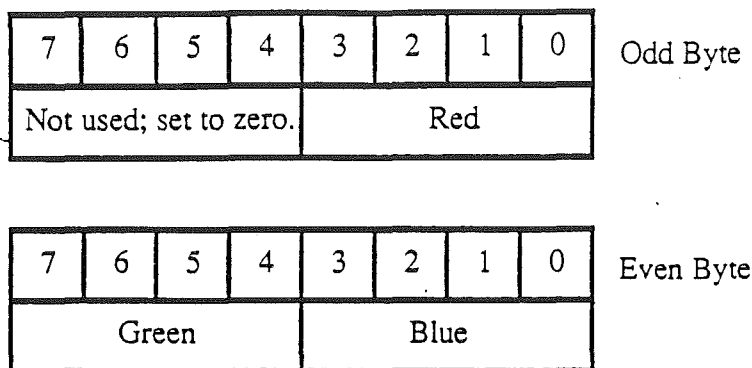


Figure 4-7. Color palette format.

These three primary colors, Red, Green and Blue (RGB), are different and not related to the four **m&m's** primary colors, Brown, Orange, Green and Yellow (BOGY). The former colors may, however, melt in your hand.

The color palettes are located in video buffer locations \$9E00-\$9FFF. Sixteen color palettes reside in this space, with 32 bytes per palette. Each color palette represents 16 colors, with two bytes per color. The palette indicated in the pointer byte is used to display the pixels in color on the scan line. The color format for the color bytes is shown in Figure 4-7. The starting address for each of the color palettes and the colors within them are listed in Table 4-2. The 16 colors within a palette have numbers \$0 through \$F. Note that each color begins on an even address.

Once you have filled the palettes with the colors to be used and selected the display modes within each of the pointer bytes, you must choose which of the 16 colors that you are going to display for each pixel.

Palette Number	Color \$0	Color \$1	...	Color \$E	Color \$F
\$0	\$9E00-01	9E02-03	...	9E1C-1D	9E1E-1F
\$1	\$9E20-21	9E22-23	...	9E3C-3D	9E3E-3F
\$2	\$9E40-41	9E42-43	...	9E5C-5D	9E5E-5F
⋮	⋮	⋮	⋮	⋮	⋮
\$F	\$9FE0-E1	9FE2-E3	...	9FFC-FD	9FFE-FF

Table 4-2. Palette and color starting addresses.

Pixels

The Super Hi-Res color information for each pixel is different for each of the two resolution modes: four bits represent each pixel color in 320-pixel mode; two bits represent the pixel color in 640-pixel mode. Higher resolution comes with a slight penalty, however:

while in 320 mode a pixel may be any of 16 colors, a pixel may be one of only 4 colors in 640 mode.

The pixel data is located in the display buffer in a linear and contiguous manner; \$2000 corresponds to the upper-left corner of the display, and \$9CFF corresponds to the lower-right corner. Each scan line uses 160 (\$A0) bytes. Figures 4-8 shows the format in which the pixel color data is stored in both the 320-pixel and 640-pixel modes.

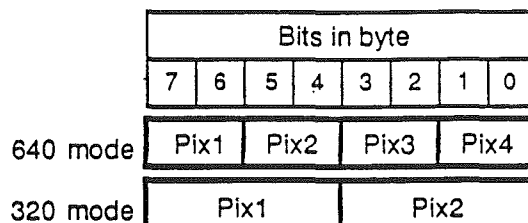


Figure 4-8. Pixel Data Byte Format

In the 320-pixel mode, four bits determine each pixel color, and data is stored two pixels to a byte of the display buffer. Since four bits determine the pixels color, in 320 mode each pixel can be any of the sixteen colors from that palette.

In 640 mode, color selection is more complicated. The 640 pixels in each horizontal line occupy 160 adjacent bytes of memory, and each byte holds four pixels that appear side-by-side on the screen. The sixteen colors in the palette are divided into four groups of four colors each. The first pixel in each horizontal line can select any one of four colors from the third group of four in the palette. The second pixel selects from the fourth group of four colors in the palette. The third pixel selects from the first group of four colors, and the fourth pixel selects from the second group, as shown in Figure 4-9. The process repeats for each successive group of four pixels in a horizontal line. Thus, even though a given pixel can be one of only four colors, different pixels in a line can take on any of the sixteen colors in a palette. Using a technique called **dithering**, software for 640 mode can take advantage of this color-selection scheme to display 16-color graphics on the same screen with 80-column text.

Dithering is a technique for alternating the values of adjacent pixels to create the effect of more colors.

Pixel	Value	Palette
Px3	0	Color1
	1	Color2
	2	Color3
	3	Color4
Px4	0	Color5
	1	Color6
	2	Color7
	3	Color8
Px1	0	Color9
	1	Color10
	2	Color11
	3	Color12
Px2	0	Color13
	1	Color14
	2	Color15
	3	Color16

Figure 4-9. Color selection in 640 mode

Color-Fill mode

Color-Fill mode, which is available only in 320-pixel mode, is used to fill rapidly a large area of the video display with a single color. In this mode, color \$0 in the palette takes on a unique definition. Any pixel data byte containing the color value \$0 causes that pixel to take on the color of the previous pixel instead of displaying a palette color. This means that only 15 unique palette colors (\$1-F) are available for each scan line rather than 16 colors. For example, assume that A, B, and C represent three different palette colors, four bits per pixel. These colors do not include color \$0. The desired color pattern for a series of pixels on a line might be as follows:

Without Color-Fill mode:

AAAAAAAAAAAAABBBBBBBBBBBBBBBBBCCCCCCCCCCCC

These pixel values would produce the same color pattern by using Color-Fill mode:

Using Color-Fill mode:

A0000000000000B0000000000000C000000000000

Method 2 would save time: the program only needs to fill the pixel area of the scan line once with 0, and then write a color value into those locations where a color should begin or change. In the example just given, only three bytes need be written to implement the three color areas on the scan line using the Color Fill method, as opposed to 12 pixels per color without Color-Fill.

The only restriction of the Color Fill mode is that the first pixel value on a scan line must not be 0; if the first pixel value is 0, then an undetermined color results.

VGC interrupts

Video display in the Apple Cortland is enhanced by VGC-generated interrupts. The VGC can handle three interrupt sources: two that are generated internally (One-Second and Scan-Line) and one externally (External). The VGC generates only the two internal interrupts; the external interrupt option is not used in this computer, and is permanently disabled.

A 1-Hz input signal from the RTC chip sets the One-Second interrupt status bit. The Scan-Line interrupt occurs at the beginning of a video display scan line which has the Generate Interrupt bit set in the corresponding pointer byte. Scan-Line interrupts are generated only when the computer is operating in the Super Hi-Res video graphics mode, and are not available in other video modes.

Figure 4-10 depicts the video screen consisting of the text display area and the display border. The Scan-Line interrupt occurs at the beginning of the scan line which is defined as the beginning of the right-hand border area as shown in figure 4-10.

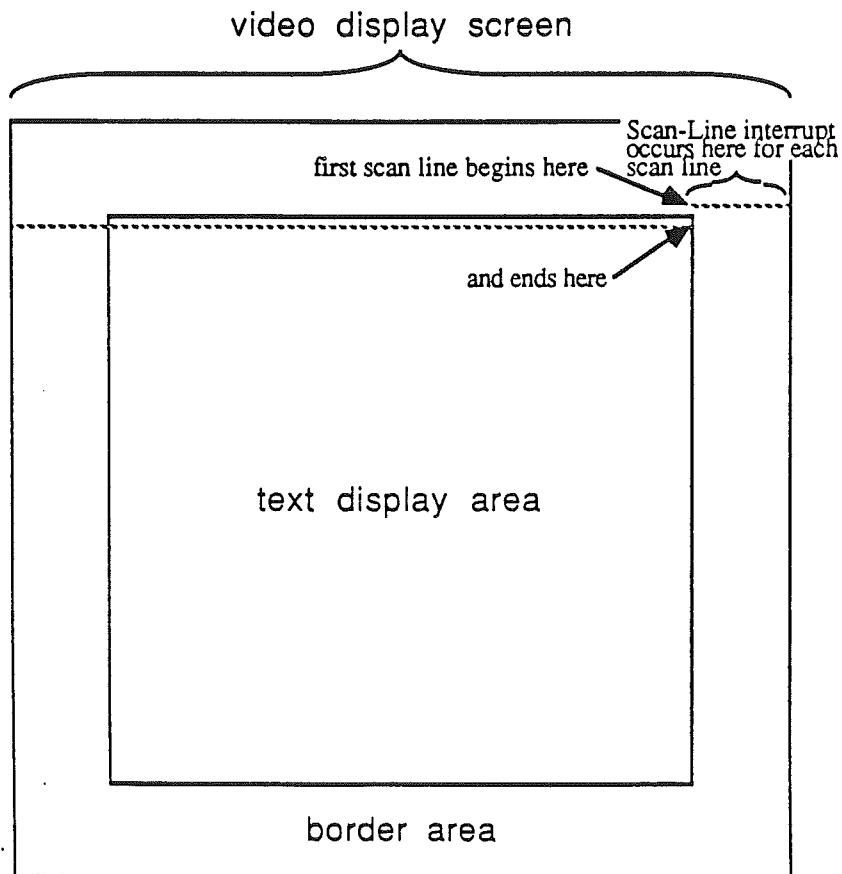


Figure 4-10. Scan-Line interrupt.

The VGC Interrupt register

The VGC Interrupt register (\$C023) contains a status bit and an enable bit for each of the three interrupts. When an interrupt occurs, the interrupt status bit for that interrupt is set.

The VGC Interrupt bit (bit 7) is set and the IRQ line is asserted if the interrupt status bit *and* interrupt enable bit are set for one or more interrupts.

To enable an interrupt by writing to the appropriate positions in the Interrupt register, the interrupt source hardware sets the status bits. Software can directly manipulate only the enable bits in the VGC Interrupt register; writing to the other bit positions has no effect. Figure 4-11 shows the format of the VGC Interrupt register and is followed by a description of each register bit.

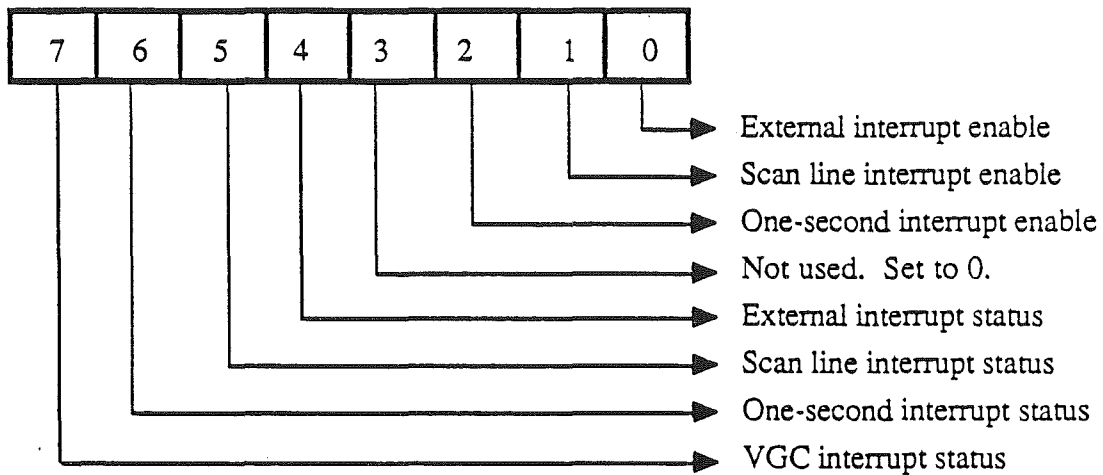


Figure 4-11. VGC interrupt register at \$C023.

Bit	Value	Description
7	1	VGC Interrupt status. This bit is set when the interrupt bit and the status bit is set for one or more of the interrupts.
	0	This bit is 0 when all interrupts have been cleared.
6	1	One-Second interrupt status. 1 = interrupt has occurred.
	0	0 = interrupt is cleared.
5	1	A Scan-Line interrupt status. 1 = interrupt has occurred.
	0	0 = interrupt is cleared.
4	-	This interrupt is permanently disabled in the Cortland. Will always be 0.
3	-	Not used. Set to 0.
2	1	One-Second interrupt enabled.
	0	Interrupt is disabled.
1	1	Scan-Line interrupt enabled.
	0	Interrupt is disabled.
0	-	This interrupt is permanently disabled in the Cortland. Set to 0.

The VGC Interrupt-Clear register

Once an interrupt has occurred, the interrupt routine must proceed to clear the interrupt and take some pre-determined interrupt-handling action. To clear the Scan-Line and One-Second status bits, write a 0 into the corresponding bit position in the Interrupt-Clear register at \$C032. Bit 5 clears the Scan-Line interrupt and bit 6 clears the One-Second interrupt in the Interrupt-Clear register shown in figure 4-12. Writing a 1 into these positions or writing into the other bit positions has no effect. Reading the video counters will also clear the Scan-Line status bit. Figure 4-12 shows the format of the VGC Interrupt-Clear register, and is followed by a description of each bit.

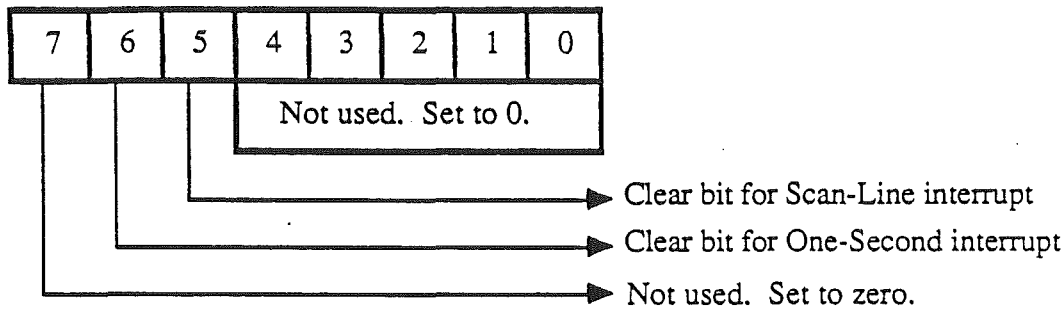


Figure 4-12. VGC interrupt clear register at \$C032.

Bit	Value	Description
7	-	Not used. Set to 0.
6	1	Undefined result.
	0	Write a 0 here to clear the One-Second interrupt.
5	1	Undefined result.
	0	Write a 0 here to clear the Scan-Line interrupt.
4-0	-	Not used. Set to 0.

Graphics Summary

The Apple Cortland supports all previous Apple II graphics modes, and provides enhancements to these modes. These are

- the ability to select unique text and background colors
- the ability to select the border color for the perimeter of the video image
- the ability to display gray-scale video

New graphics modes including

- Super Hi-Res graphics mode in 320-pixel resolution
- Super Hi-Res graphics mode in 640-pixel resolution
- Color-Fill mode

Sixteen palettes, each palette containing 16 preselected colors, are located in the palette area of the display buffer. Use these palettes to select the display colors for each pixel

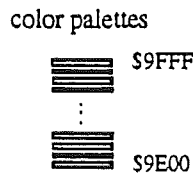
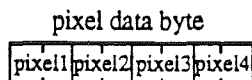
200 pointer bytes (one pointer byte per scan line) which determine

- either 320-pixel mode or 640-pixel mode
- color-fill mode or regular display mode
- which of the 16 palettes in memory are to be used in this scan line
- whether the current scan line interrupt is enabled or disabled

The pixel data bytes are then loaded with the color information for each pixel: there will be four bits per pixel in 320 mode, and there will be two bits per pixel in 640 mode.

Figure 4-13 shows the display screen and the pixels which make up each scan line. Also shown are the pixel data bytes for both 640- and 32-pixel Super Hi-Res graphics mode. The pointer bytes, one for each scan line are shown at the right.

640-pixel mode:



320-pixel mode:

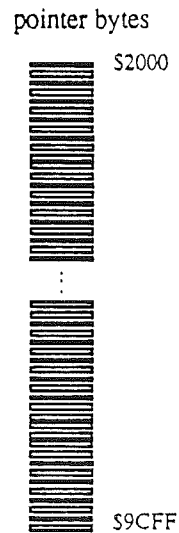
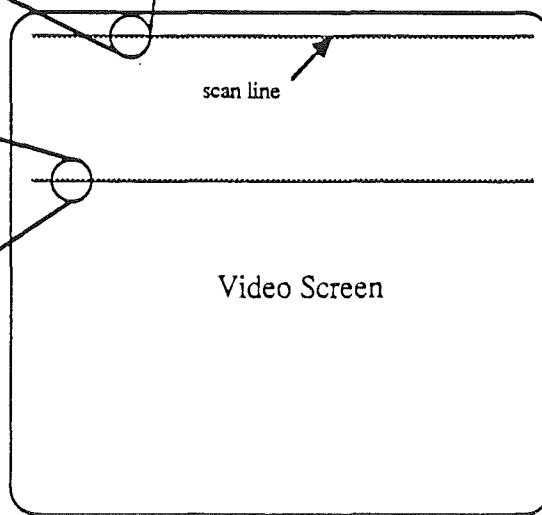
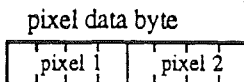


Figure 4-13. Drawing pixels on the screen.

Chapter 5

Peripheral Expansion

Introduction

The main circuit board of the Apple Cortland has seven empty card connectors or slots on it. These slots make it possible to add features to the Apple Cortland by plugging in peripheral cards with additional hardware. This chapter describes the hardware that supports these slots, including the signals available at the expansion slots. Figure 5-1 shows a block diagram of the Apple Cortland and the relationship of the slots in the computer.

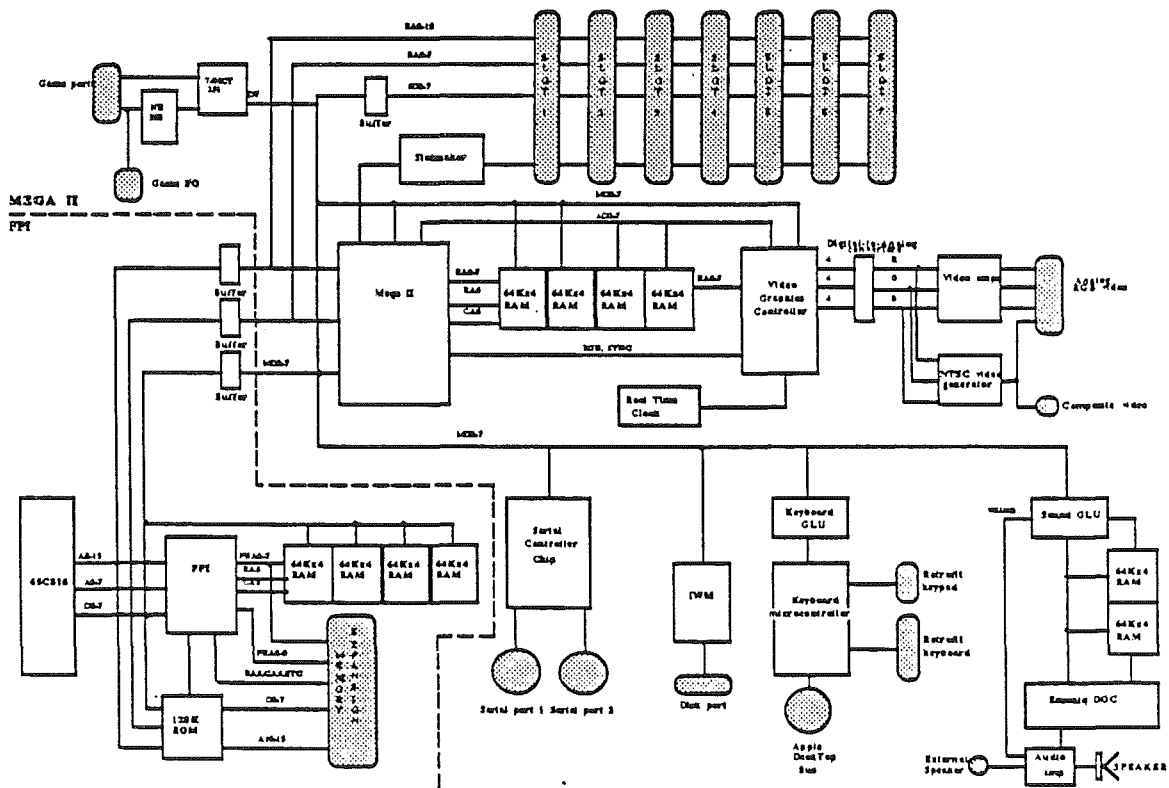


Figure 5-1. Cortland block diagram.

Note: The Apple Cortland has seven expansion slots plus a memory expansion slot. This memory expansion slot is not the same as the seven expansion slots, nor should it be used as such. Also, the memory expansion slot is not the same as the

auxiliary slot in the Apple //e, nor should it be used as such. The memory expansion slot is to be used for memory expansion cards designed specifically for this slot. See the *Memory Expansion* chapter in this manual for a description of this slot.

The Expansion slots

The seven connectors lined up across the back part of the Apple Cortland main circuit card are the expansion slots, also called peripheral slots or simply slots, numbered from 1 to 7. They are 50-pin card-edge connectors with pins on 0.10-inch centers. A circuit card plugged into one of these connectors has access to all of the signals necessary to perform input and output and to execute programs in RAM or ROM on the card. These signals are shown in figure 5-2 and are described briefly in Table 5-1.

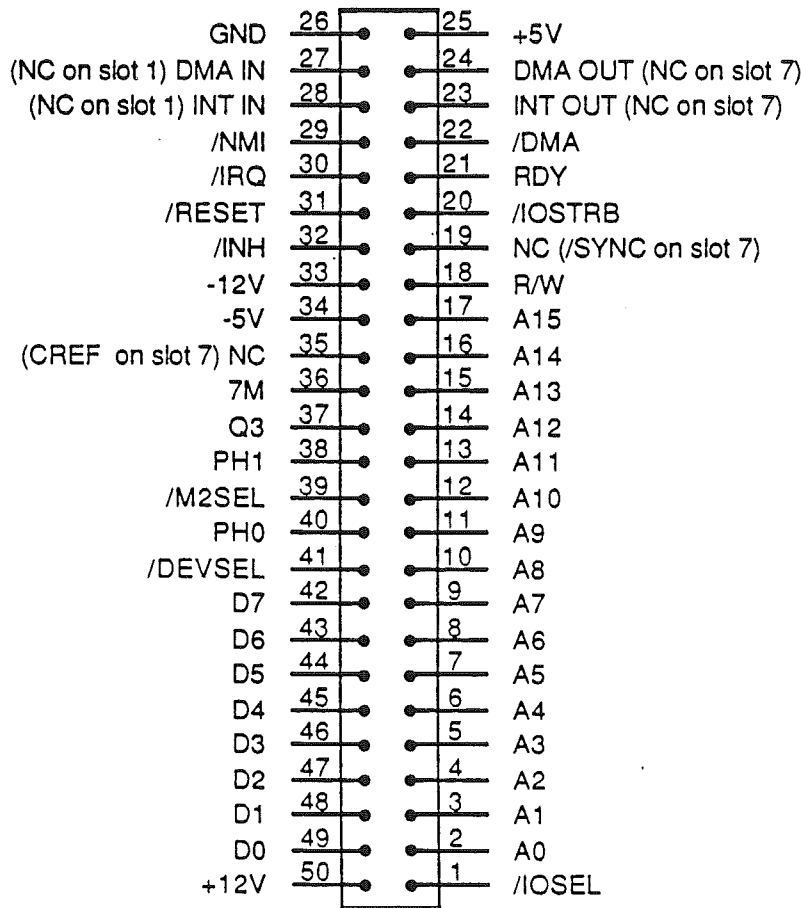


Figure 5-2. Input/Output Pinout Diagram

Table 5-1
Expansion slot signals

Pin	Signal	Description
1	I/O SELECT	Normally high; goes low during $\phi 0$ when the 65816 addresses location $\$CnXX$, where n is the connector number. This line can drive 10 LS TTL loads.*
2-17	A0-A15	Three-state address bus. The address becomes valid during $\phi 1$ and remains valid during $\phi 0$. Each address line can drive 2 LS TTL loads.*
18	R/W'	Three-state read/write line. Valid at the same time as the address bus; high during a read cycle, low during a write cycle. It can drive 2 LS TTL loads.*
19	SYNC ϕ	Composite horizontal and vertical sync, on expansion slot 7 only. This line can drive 2 LS TTL loads.*
20	I/O STROBE'	Normally high; goes low during $\phi 0$ when the 65816 addresses a location between $\$C800$ and $\$CFFF$. This line can drive 4 LS TTL loads.
21	RDY	Input to the 65816. Pulling this line low during $\phi 1$ halts the 65816 with the address bus holding the address of the location currently being fetched. This line has a 4700 ohm pullup resistor to +5V.
22	DMA'	Input to the address bus buffers. Pulling this line low during $\phi 1$ disconnects the 65816 from the address bus. This line has a 3300 ohm pullup resistor to +5V.
23	INT OUT	Interrupt priority daisy-chain output. Usually connected to pin 28 (INT IN).†
24	DMA OUT	DMA priority daisy-chain output. Usually connected to pin 22 (DMA IN).
25	+5V	+5-volt power supply. A total of 500mA is available for all peripheral cards.
26	GND	System common ground.
27	DMA IN	DMA priority daisy-chain input. Usually connected to pin 24 (DMA OUT).
28	INT IN	Interrupt priority daisy-chain input. Usually connected to pin 23 (INT OUT).

Cortland Hardware Reference Manual

29	NMI	Non-maskable interrupt to 65816. Pulling this line low starts an interrupt cycle with the interrupt-handling routine at location \$03FB. This line has a 3300 ohm pullup resistor to +5V.
30	IRQ	Interrupt request to 65816. Pulling this line low starts an interrupt cycle only if the interrupt-disable (I) flag in the 65816 is not set. Uses the interrupt-handling routine at location \$03FE. This line has a 3300 ohm pullup resistor to +5V.
31	RES	Pulling this line low initiates a reset routine.
32	INH	Pulling this line low during $\phi 1$ inhibits (disables) the memory on the main circuit board. This line has a 3300 ohm pullup resistor to +5V.
33	-12V	-12 volt power supply. A total of 200mA is available for all peripheral cards.
34	-5V	-5 volt power supply. A total of 200mA is available for all peripheral cards.
35	CREF	3.58 MHz color reference signal, on slot 7 <i>only</i> . This line can drive 2 LS TTL loads.*
35	M2B0	Mega II bank-0 signal, on slot 1 <i>only</i> . This signal goes low whenever the Mega II is addressing the main bank of Mega II RAM.
36	7M	System 7 MHz clock. This line can drive 2 LS TTL loads.*
37	Q3	System 2 MHz asymmetrical clock. This line can drive 2 LS TTL loads.*
38	$\Phi 1$	phase-1 clock. This line can drive 2 LS TTL loads.*
39	M2SEL	The Mega II select signal. This signal goes low whenever the Mega II is addressing a location within the 128K of Mega II RAM.
40	$\Phi 0$	phase-0 clock. This line can drive 2 LS TTL loads.*
41	DEVICE SELECT	Normally high; goes low during $\Phi 0$ when the 65816 addresses location \$C0nX, where <i>n</i> is the connector number plus 8. This line can drive 10 LS TTL loads.*

42-49 D0-D7	Three-state buffered bi-directional data bus. Data becomes valid during $\Phi 0$ high and remains valid until $\Phi 0$ goes low. Each data line can drive one LS TTL load.*
50 +12V	+12 volt power supply. A total of 250mA is available for all peripheral cards.

* Loading limits are for each card.

† On slot 7 only, this pin can be connected to the graphics-mode signal GR: see text for details.

Apple II compatibility

The Cortland seven I/O slots are almost identical to the slots in the Apple II, the only exceptions being M2B0 on pin 35 (at slot 1 only) and /M2SEL which replaces μ PSYNC on pin 39. The slots behave like their counterparts in the Apple II with only a few differences, the most important one being the behavior of the address bus. Since the Cortland computer can operate at 2.8 MHz and has a 24-bit address, the address bus to the slots is not always valid as it was in the Apple II. The signal /M2SEL indicates when a valid address for banks 224 or 225 (\$E0 or \$E1) is present on the address bus and so should be used to qualify any address decoding that does not use one of the I/O enable lines. Since this memory space contains video buffers and I/O addresses, peripheral video cards can make extensive use of these two signals.

Direct Memory Access

Direct Memory Access (DMA) supports the full 24-bit address range. This means that any peripheral card using DMA may have direct address control of all 8 megabytes of memory (main and expansion memory). This is accomplished loading the DMA bank register with the upper eight bits of the required 24-bit address.

During DMA cycles (memory access cycles that are controlled by a DMA peripheral card), the address bus is turned off until the bank address has been latched. At this time, the address bus is enabled, pointing "in" toward the FPI and 65C816. The FPI decodes the address and stored DMA bank address to determine whether the cycle is to RAM, ROM, or Mega II. If the cycle is a DMA to the Mega II (or slots), the Mega II select line is asserted by the FPI and the FPI data buffers are turned off if R/W is high. If the access is to the hi-speed RAM, the data buffers are enabled while PH0 is high.

Note: To increase read/write data timing margins to the hi-speed RAMs, the FPI generates an early CAS signal for read cycles and a late CAS signal for write cycles. This provides earlier read data available and less required write data setup time.

Interfacing the Cortland System

The input and output functions are made possible by built-in I/O devices and the use of peripheral, slot I/O and DMA cards.

Slot I/O Cards

Most I/O cards used in the Apple II also work in the Cortland system. Cards that use the IOSEL and DEVSEL bus signals will not be confused by the larger address range in the Cortland system.

The 65816 CPU operates with a 24-bit address; however, the I/O slots receive only a 16-bit address. Therefore, cards that use the 16-bit address decode select method rather than the DEVSEL and IOSEL signals will not work properly. These cards include the multi-function I/O cards that emulate multiple I/O cards, and most add-on RAM cards. In general, these type of cards will not be needed because of the extensive built-in I/O and hi-speed RAM expansion already provided.

Cards that use INHIBIT will work properly if:

- a) the system is running at 1 MHz, and
- b) they assert inhibit within 200 ns of the PH0 falling edge.

However, compatibility with this type of card must be determined on an individual basis because many Monitor calls execute code in bank \$FF and the many cards are not designed to decode bank information.

The FPI will ignore INHIBITs that occur when the system is running fast (2.5 MHz), or when it is not in a bank where I/O and language-card operation is enabled. This improves compatibility with existing cards.

DMA Cards

Many DMA cards that work successfully in previous Apple II models will work in the Cortland system, but may require changes in their firmware or associated software to function properly with the DMA bank register. In general, DMA cards that assert/remove the DMA signal within the first 200 ns of the PH0 rising edge will probably work properly; this allows sufficient time for the Mega II select line to be activated by the FPI when video and I/O accesses are required.

Note: Normally the system should be running slowly when performing DMA, otherwise, DMA to I/O or Mega II video areas will not work properly. However, DMA can be performed while the system is running fast as long as the following constraints are observed:

Note: Only hi-speed RAM or ROM is accessed (access to I/O, video, or the Mega II banks do not work properly).

Note: Fast DMA may cause a repeated cycle to occur to the location currently being accessed by the processor. This could cause a malfunction if the CPU is accessing I/O when the DMA occurs; however, a repeated access to a RAM or ROM location will have no effect.

The 65C816 can be stopped indefinitely for DMA and does not require any CPU refresh cycles from a DMA card.

Expansion Slot Signals

Many of the expansion slot signals can be grouped into three general categories:

- those that constitute and support the address bus
- those that constitute and support the data bus
- those that support the functions of DMA and interrupts

These signals are described in the following paragraphs. For additional information, refer to the schematic diagram in Appendix E.

The buffered address bus

The microprocessor's address bus is buffered by two 74HCT245 octal three-state bi-directional buffers. The 65C816 R/W line is also buffered. The FPI disables these buffers when requested by any peripheral card. This disables the address and R/W lines so that peripheral DMA circuitry can control the address bus. The DMA address and R/W signals supplied by a peripheral card must be stable all during $\Phi 0$ of the instruction cycle, as shown in Figure 5-3.

Another signal that can be used to disable normal operation of the Apple Cortland is INH'. Pulling INH' low disables all of the memory in the Apple Cortland except the part in the I/O space from \$C000 to \$CFFF. A peripheral card that uses either INH' or DMA' must observe proper timing; in order to disable RAM and ROM properly, the disabling signal must be stable all during $\Phi 0$ of the instruction cycle (refer to the timing diagram in Figure 5-3).

The peripheral devices should use I/O SELECT' and DEVICE SELECT' as enables. Most peripheral ICs require their enable signals to be present for a certain length of time before data is strobed into or out of the device. Remember that I/O SELECT' and DEVICE SELECT' are only asserted during $\Phi 0$ high.

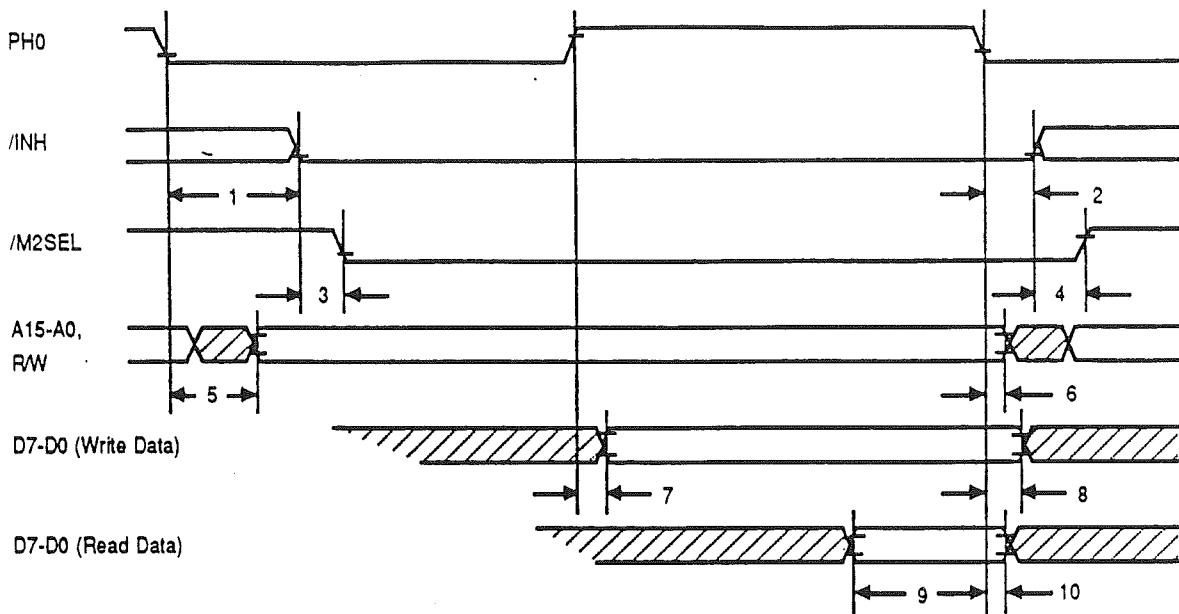


Figure 5-3. I/O Read and Write Timing with /INH Active

Number	Description	Min(ns)	Max(ns)
1	/INH valid after PH0 low		175
2	/INH hold time	15	
3	/INH low to /M2SEL low delay		30
4	/INH high to /M2SEL high delay		30
5	Address and R/W valid from PH0 low		100
6	Address and R/W hold time	15	
7	Write data valid delay		30
8	Write data hold time	30	
9	Read data setup time to PH0	140	
10	Read data hold time	10	

Read and write cycles that are directed to the I/O slots by /INH have the same timing parameters as normal I/O read and write cycles. When /INH is asserted, the computer responds as if a Mega II memory cycle were being performed.

Cards that use the /INH signal will function properly only if the computer is running at slow speed (1 MHz). If the computer is running at high speed, the addresses that are seen by cards in the I/O slots are not guaranteed to be valid during an entire PH0 cycle. Also, since the upper 8 bits of the memory address are not available to cards (only 16 address lines are available at the slots), the potential of /INH is greatly reduced in this machine.

The slot data bus

The Apple Cortland has three versions of the microprocessor data bus:

- the internal bus DBUS, connected directly to the microprocessor and the FPI chip and all main RAM
- the Mega II bus MDBUS, connecting the Mega II, VGC, SCC, IWM, Keyboard and Sound GLUs and the Mega II RAM main bank
- the slot data bus, SDBUS, common to all expansion slots

The 65816 is fabricated with MOS circuitry, so it can drive capacitive loads of up to about 130 pF. If peripheral cards are installed in all seven slots, the loading on the data bus can be as high as 500 pF, so the 74245 buffer is used to drive the data bus peripheral card loads. The same argument applies if you use MOS devices on peripheral cards: they don't have enough drive for the fully-loaded bus, so you should add buffers. A peripheral card must have the capacity to drive 2 LSTTL loads per slot-pin, plus additional capacitance for the Cortland data bus.

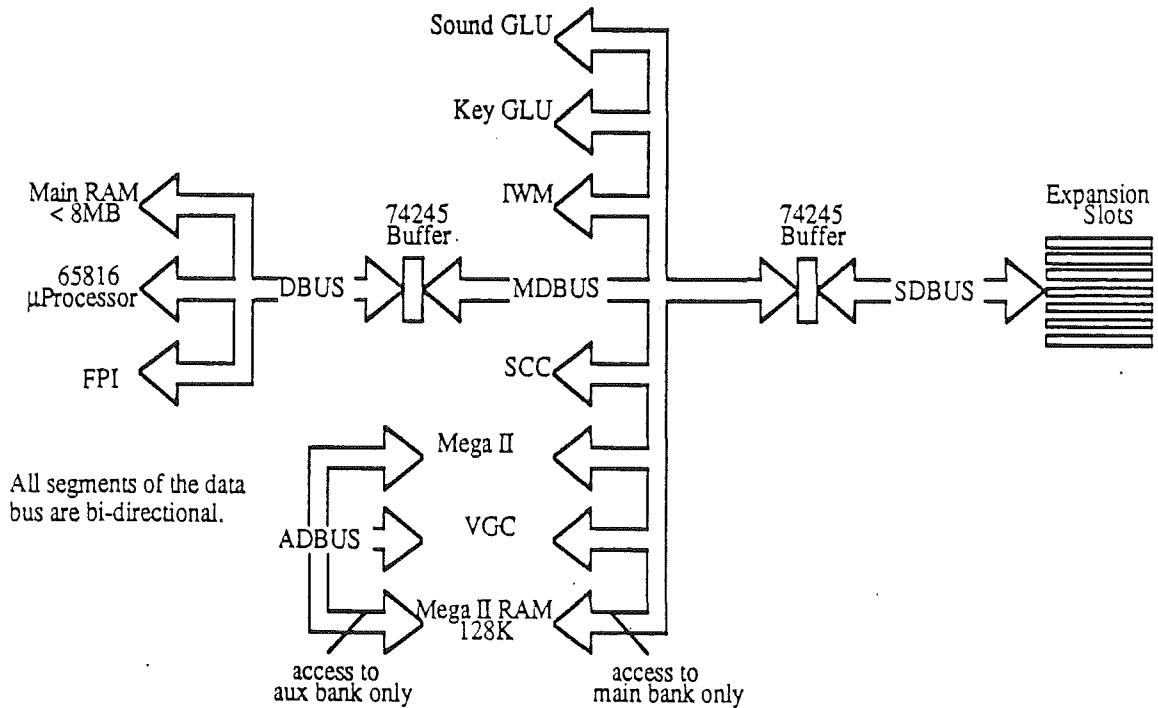


Figure 5-4. The data buses within the Cortland.

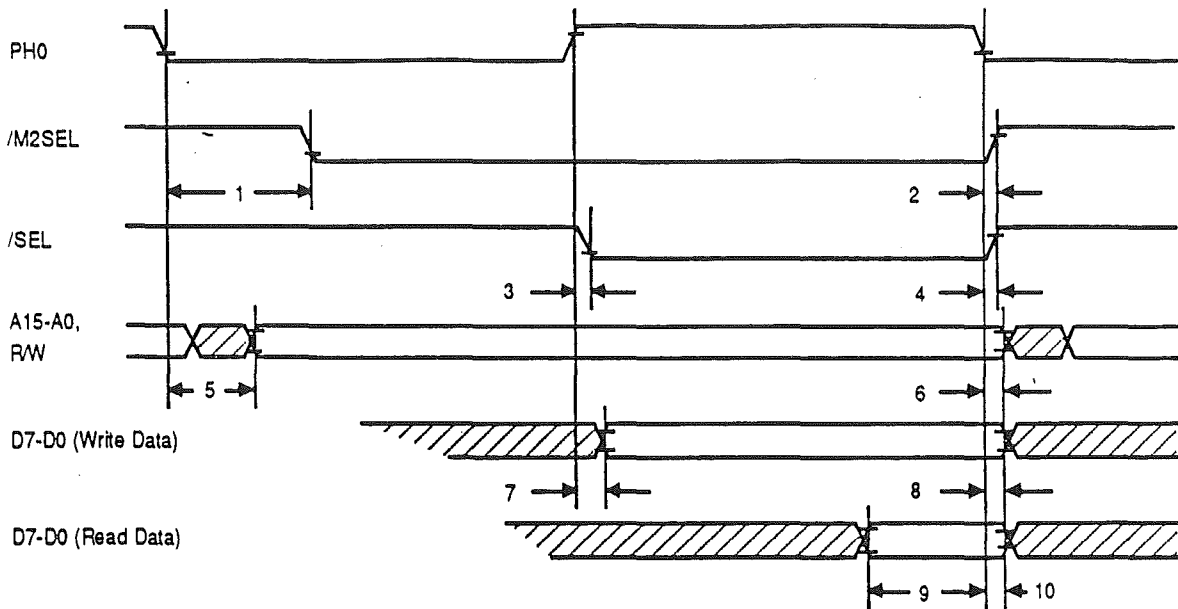


Figure 5-5. Slot I/O Read and Write timing.

Number	Description	Min(ns)	Max(ns)
1	/M2SEL low from PH0 low		160
2	/M2SEL hold time	-10	
3	I/O enable low from PH0 high		15
4	(DEV _n /IOSEL _n /IOSTRB) I/O enable high from PH0 low	10	
5	(DEV _n /IOSEL _n /IOSTRB) Address and R/W valid from PH0 low		100
6	Address and R/W hold time	15	
7	Write data valid delay		30
8	Write data hold time	30	
9	Read data setup time to PH0	140	
10	Read data hold time	10	

The standard Cortland slot I/O timing is shown in Figure 5-5. When the Cortland computer is running in high-speed mode (2.8 MHz), the address bus to the I/O slots is not valid during the entire PH0 cycle, and therefore cannot be used to perform unqualified address decoding. The /M2SEL signal (which replaces the μ SYNC signal found in previous Apple II models), indicates when a slow, synchronized memory cycle is taking place and therefore, when the value on the address bus will remain valid during the current PH0 cycle. This means that cards that use the Apple II technique of "phantom

slotting" to put multiple I/O devices on one card must use /M2SEL to qualify their address decoding.

Interrupt and DMA daisy chains

The interrupt requests (IRQ' and NMI') and the direct-memory access (DMA') signal are available at all seven expansion slots. A peripheral card requests an interrupt or a DMA transfer by pulling the appropriate output line (pin 24) low. If two peripheral cards request an interrupt or a DMA transfer at the same time, they will contend for the data and address busses. To prevent this, two pairs of pins on each connector are wired as a priority daisy chain. The daisy-chain pins for interrupts are INT IN (pin 28) and INT OUT (pin 23), and the pins for DMA are DMA IN (pin 27) and DMA OUT (pin 24), as shown in Figure 5-2.

Each daisy chain works like this: the output from each connector goes to the input of the next higher numbered one. For these signals to be useful for cards in lower numbered connectors, all of the higher numbered connectors must have cards in them, and all of those cards must connect DMA IN to DMA OUT and INT IN to INT OUT. Whenever a peripheral card uses pin DMA', it must do so only if its DMA IN line is active, and it must disable its DMA OUT line while it is using DMA'. The INT IN and INT OUT lines must be used the same way: enable the card's interrupt circuits with INT IN, and disable INT OUT whenever IRQ' or NMI' is being used.

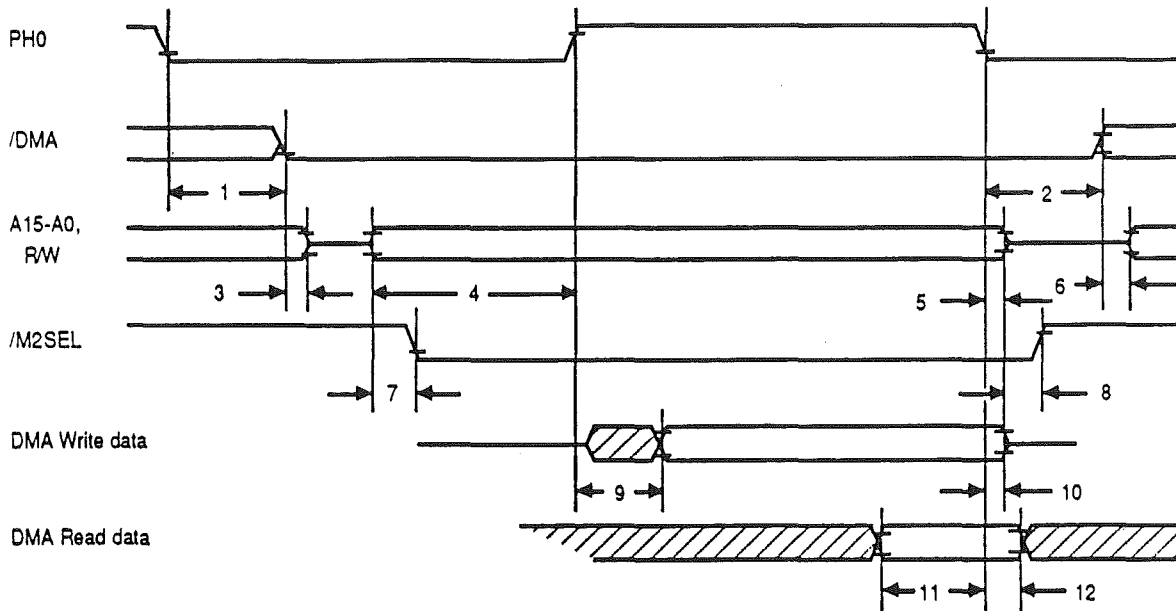


Figure 5-6. /DMA Read and Write Timing

Number	Description	Min(ns)	Max(ns)
1	/DMA low from PH0 low		120
2	/DMA high from PH0 low		120
3	A15-A0 and R/W float from /DMA		30
4	DMA address and R/W valid before PH0 goes high	300	
5	DMA address and R/W hold time after PH0 goes high	10	
6	/DMA high to A15-A0 and R/W active		30
7	DMA address valid to /M2SEL low		30
8	DMA address float to /M2SEL high		30
9	PH0 high to write data valid		100
10	DMA write data hold time	10	
11	DMA read data setup time	125	
12	DMA read data hold time	30	

DMA devices will work in the Cortland computer only in slow mode (1 MHz). If the computer is running at high speed (2.8 MHz), only DMA accesses to the high-speed memory banks 0 through 127 will work. Accesses to the low-speed memory (all I/O and video memory) must be done at low speed (1 MHz). To do this, set the processor speed bit in the configuration register at location \$XXXX before requesting DMA.

DMA can be performed to or from any part of the Cortland memory map, provided that the DMA Bank Register (\$XXXX) is first set to the appropriate bank.

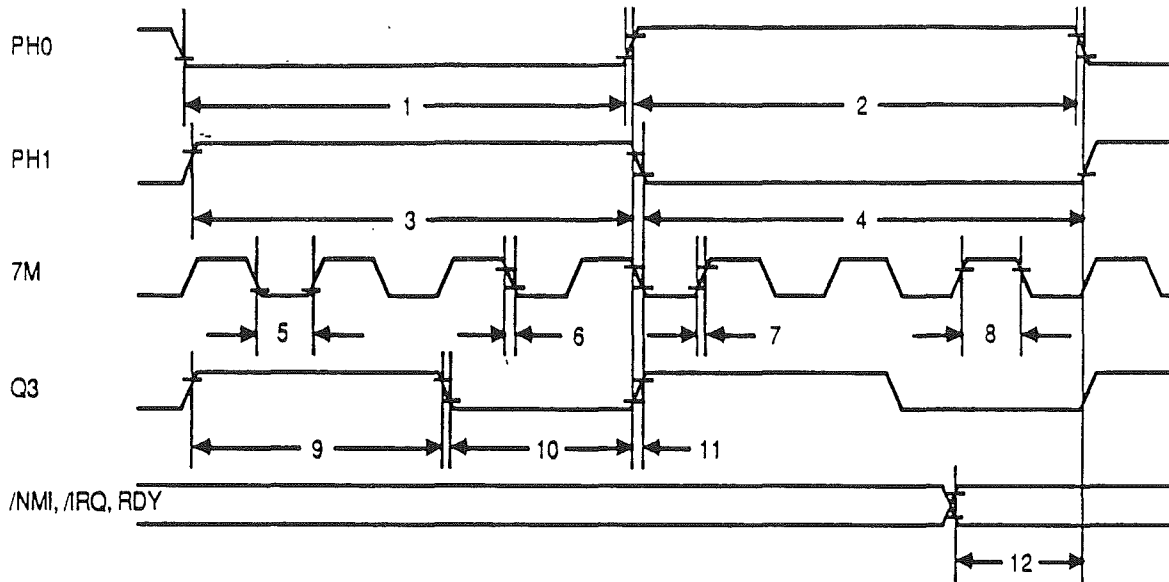


Figure 5-7. Input/Output Clock and Control Signal Timing

Number	Description	Min(ns)	Max(ns)
1	PH0 low time	480	
2	PH0 high time	480	
3	PH1 high time	480	
4	PH1 low time	480	
5	7M low time	60	
6	Fall time, all clocks		10
7	Rise time, all clocks		10
8	7M high time	60	
9	Q3 high time	270	
10	Q3 low time	200	
11	Skew, PH0 to other clock signals	-10	10
12	Control signal setup time	140	

Note: All clock signals present on the I/O slots are buffered by the Slotmaker custom I.C. These clock signals are delayed somewhat from the corresponding signals on the main board because of this buffering. All timing parameters in the timing diagrams in this chapter have been adjusted to account for this delay.

Loading and driving rules

Table 5-2 shows the drive requirements and loading limits for each pin on the expansion slots. The address bus, the data bus, and the R/W' line should be driven by three-state buffers. Remember that there is considerable distributed capacitance on these busses and that you should plan on tolerating the added load of up to six additional peripheral cards. MOS devices such as PIAs and ACIAs cannot switch such heavy capacitive loads; connecting such devices directly to the bus will lead to possible timing and level errors. Buffer all MOS output signals.

The total power supply current available for all seven expansion slots is

- 500 mA at +5V
- 250 mA at +12V
- 200 mA at -5V
- 200 mA at -12V.

The support circuitry for the slots is designed to handle a DC load of 2 LS TTL loads per slot pin and an AC load of no more than 15 pF per slot pin.

Summary

- The Apple Cortland expansion slots are almost identical to other Apple II expansion slots. The exceptions, signals M2B0 and M2SEL indicate accesses to slow RAM banks \$E0 and \$E1, the location of I/O and video buffers.
- Expansion slot outputs are buffered to provide greater driving capability. Peripheral card must use buffers when drive buses on the Cortland.
- The power supply provides power the peripheral cards. This power is limited and must not be exceeded.
- The Apple Cortland expansion slots are provided to allow an external device a means to connect to the internal data and address buses. DMA and interrupt requests are handled in a slot-7-to-slot-1 priority fashion. A card in the higher-numbered slot has priority when more than one device signals a request simultaneously.

Chapter 6

Cortland Sound

Introduction

One of this computer's features is the outstanding sound capability. By programming the Apple Cortland you can utilize this powerful sound-synthesizing capability, your ability to generate sounds is limited only by your imagination. This chapter covers the Digital Oscillator Chip (DOC), the individual oscillators and the many registers associated with these oscillators. Also covered is the sound General Logic Unit (GLU) and the registers associated with the GLU.

Sound Synthesis

Synthesized sound is the process of programming digital oscillators to produce waveforms simulating sounds (musical, human or other) or generating unique ones. These waveforms can be programmed manually (values placed in memory individually) or through digitization of an outside analog input signal.

The Cortland uses the Ensoniq 5503 Digital Oscillator Chip (DOC), a programmable sound synthesizer chip. This chip has 15 independent voices, volume control, and digitizing capability. The synthesizer uses 64K of dedicated sound RAM and interfaces with the 65C816 microprocessor via the Sound General Logic Unit (GLU) chip. Commands and data are transferred to the DOC via the GLU, and sound output is output via the built-in speaker, external speaker jack, or molex connector on the motherboard. Figure 6-1 shows the relationship of the sound components to the rest of the computer.

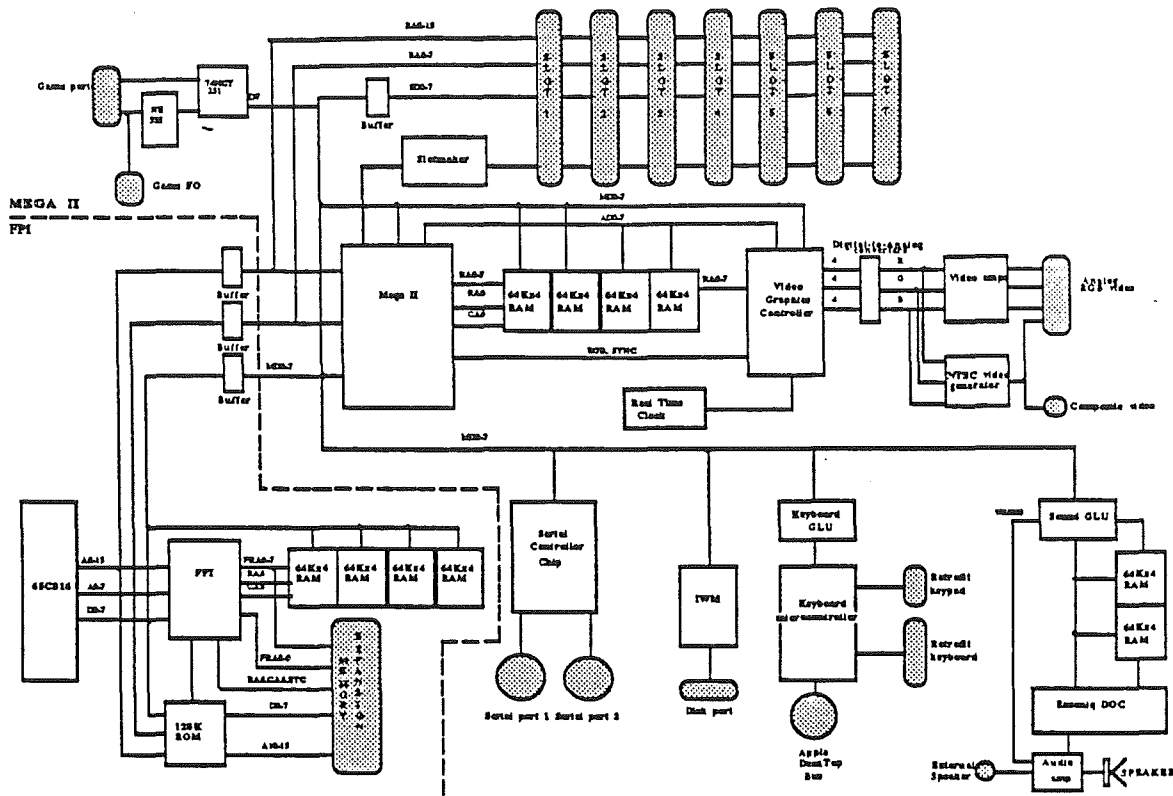


Figure 6-1. Block diagram showing relationship of the sound components.

As stated earlier, the Cortland uses a toolbox of utilities to perform many different functions: graphics, disk access and sound. The following description of the DOC is meant to familiarize you with the general principles of Cortland sound generation. When you program the Cortland for sound, using the toolbox utilities will result in the proper use of the DOC and ensure software compatibility with future Apple II products.

Sidebar:

To find out how to use the sound tools, refer to the *Cortland SoundTools* manual.

Accessing the DOC

To program the DOC or build a **wavetable** in the sound RAM you must write command and data bytes to registers within the chip. This process is facilitated by the GLU. This chip serves as an interface between the microprocessor, the DOC, and the dedicated 64K-by-8 dynamic RAM. This interface allows the DOC chip to run independent of the rest of the system.

Sidebar:

A wavetable is a contiguous portion of memory containing bytes to be used as data for the DOC which will result in sound output as represented by these data bytes.

The Sound GLU contains

- a Sound Control register
- a Data register
- a pair of Address Pointer (high and low address) registers

These registers and their addresses are listed in table 6-1.

GLU Register	Address	Type
Sound Control register	\$C03C	R/W
Data register	C03D	R/W
Address pointer, low byte	C03E	R/W
Address pointer, high byte	C03F	R/W

Table 6-1. GLU registers.

The Sound Control register

The Sound Control register controls whether the microprocessor accesses the DOC internal registers or the Sound RAM. This register also controls whether or not the Address pointers auto-increment (increment automatically after every RAM read or write, thereby avoiding the necessity of reloading the pointers with addresses after each access). Figure 6-2 shows the format of the Sound Control register, followed by a description of each bit.

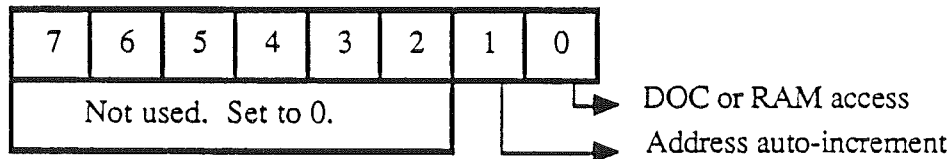


Figure 6-2. The Sound Control register at \$C03C.

Bit	Value	Description
7-2	-	Not used. Set to 0.
1	1	address auto-incrementing is enabled.
	0	address auto-incrementing is disabled.
0	1	all accesses are to the dedicated 64K RAM.
	0	all accesses are to the DOC chip.

Address Pointer register

When accessing the DOC (bit 0 = 0 in the Sound Control register), the Address Pointer register points to the address of the next byte in sound RAM. The high byte contains the high eight bits of the 16-bit address, and the low byte contains the low eight bits.

When accessing the sound RAM (bit 0 = 1 in the Sound Control register), the Address Pointer register high byte is ignored by the DOC, and the low byte points to the DOC register to be written or read from. Figure 6-3 shows the format of the Address Pointer registers.

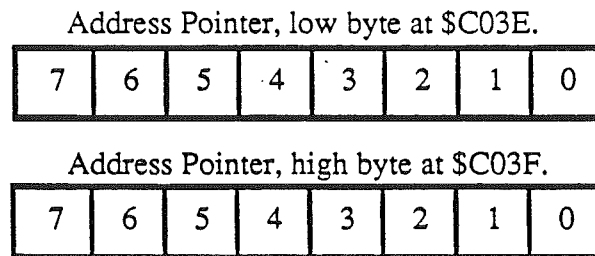


Figure 6-3. The Address Pointer registers.

Write operation

To write to the DOC or sound RAM

1. Set the Sound Control register
 - to point to either the RAM or the DOC and
 - to enable or disable auto-address incrementing
2. Then load the address pointer with the beginning location into which data is to be written
3. Write data into the corresponding memory (if you are accessing RAM) or DOC register (if you are accessing the DOC)

If the auto-increment feature is enabled, the address register is automatically incremented to the next higher location or register after each write to the data register.

Read operation

The Sound RAM read operation is the same as the write operation with one exception—reading from the data register lags by one read cycle. For example, if you want to read 10 bytes from the sound RAM, select the RAM by setting the control register bit and enabling auto incrementing. Then set the address pointer to the starting address and read the data register eleven times, discarding the first byte read.

Playing back the sound

The DOC contains 32 oscillators that, when combined in pairs (which most functions require), result in 16 voices. Digitized soundwaves are built using consecutive data bytes (known collectively as a wavetable) in dedicated sound RAM. Each oscillator reads these

bytes in sequential order at a speed that is programmable. This speed determines the frequency at which the waveform is reproduced, while the actual data in RAM determines the shape of the output waveform. The volume for each oscillator is also programmable.

The DOC registers

The DOC contains three registers common to all oscillators. These are

- the Oscillator Interrupt register
- the Oscillator Enable register
- the Analog-to-Digital (A/D) converter register

Also, each oscillator has one of each of these registers dedicated to it:

- an Oscillator Control register
- an Oscillator Data register
- a Volume register
- a Frequency register (low)
- a Frequency register (high)
- a Wavetable register
- an Address Pointer register (high)

The Oscillator Interrupt register (\$E0)

This register contains the status of the DOC Interrupt Request (IRQ) pin and the number of the oscillator that generated the interrupt, if any. When an oscillator reaches the end of a wavetable and the Enable Interrupt (EI) bit for that oscillator has previously been set, the IRQ line and bit 7 of the Interrupt register is then set, and the register number is entered in bits 1-5 of the Interrupt register. Figure 6-4 shows the format of the Interrupt register, followed by a description of each of the bits.

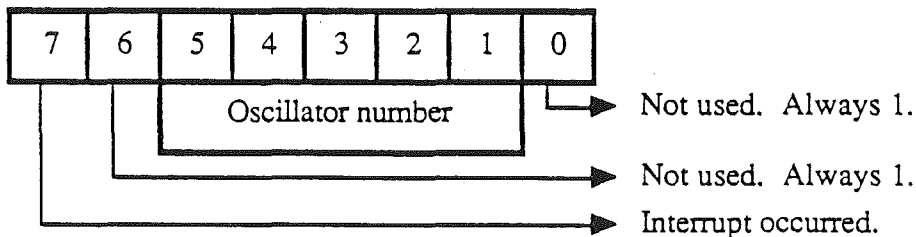


Figure 6-4. The Oscillator Interrupt register at \$E0.

Bit	Value	Description
7	1	One of the 32 DOC oscillators has generated an interrupt; this bit reflects the status of the IRQ line.
	0	No oscillator has generated an interrupt.
	-	Not used; always 1.

The Data register (\$60-7F)

The data register is a read-only register and contains the last byte read by the oscillator from the wavetable.

The Volume register (\$40-5F)

The Volume register contains the oscillator's volume value. The current wavetable data byte is multiplied by the eight-bit value to obtain the oscillator final output level.

The Frequency High and Frequency Low registers (\$00-3F)

The Frequency High register and Frequency Low register are concatenated to create a 16-bit value. This frequency value determines the speed at which the wavetable is read from memory. This indirectly determines the frequency of the output signal at the speaker. The relationship between output signal frequency, wavetable scan rate, and the Frequency register values is

$$\text{Output Frequency} = [\text{SR} / 2^{(17+\text{RES})}] * F_{\text{HL}}$$

where

$$\text{the scan rate (SR)} = 895\text{KHz} / ((\text{OSC}+2) * 8)$$

and where RES is the resolution value in the Waveform register, F_{HL} is the 16-bit frequency value concatenated from the Frequency High and Frequency Low registers and OSC is the number of enabled oscillators.

The DOC scans a waveform in memory one byte at a time. The resolution with which it scans is determined by the Oscillator Control register: a resolution of \$7 instructs the DOC to read every byte in the wavetable and place it in the data register for that oscillator. A resolution of \$6 instructs the DOC to read every other byte, \$5 to scan every third byte, and so on.

The 32 oscillators are time-domain multiplexed. That is, the DOC services each oscillator in its turn. With all oscillators enabled, the DOC takes approximately 38 microseconds to service all 32. This is a rate of 26,320 cycles per second.

As a general rule, using the same values for the address bus resolution and wavetable size (Waveform register bits 0-2 and bits 3-5 respectively) results in a linear relationship between output signal frequency and the rate at which the waveform in memory is scanned.

The Waveform register (\$C0-DF)

The Waveform register controls the size of the individual wavetable each oscillator will access. The size of this table may be between a minimum of 256 bytes and a maximum of 2 K. Figure 6-6 shows the format of the Waveform register and is followed by a description of each bit.

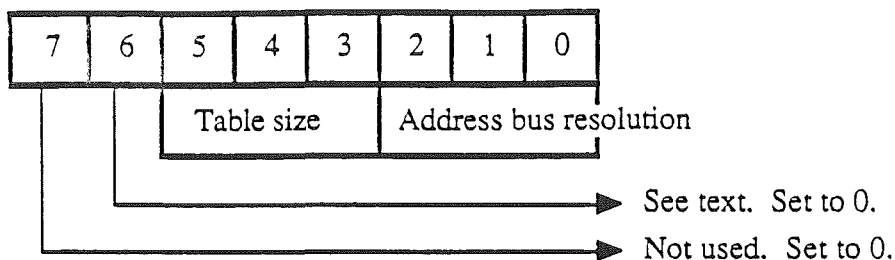


Figure 6-6. The Waveform register format.

Bit	Value	Description
7	-	Not used.
6	0	Extended addressing. The Apple Cortland uses only 64K for the sound RAM and has no high memory bank available. Therefore, this bit must always be set to 0.
5-3	-	Table size; the waveform table may extend up to 32 K-bytes in size, but in discrete steps only, as listed in the table below. Wavetables must begin on a page boundary (\$0C00, 0D00, etc.).

Bit: 5	4	3	Table size
0	0	0	256
0	0	1	512
0	1	0	1024
0	1	1	2048
1	0	0	4096
1	0	1	8192
1	1	0	16384
1	1	1	32768

Unused locations within the wavetable should begin with a minimum of eight zeroes. Otherwise, the oscillator will halt when it encounters these bytes and will not interpret them as data.

2-0	-	Address bus resolution: the wavetable may be one of eight sizes, as just shown. The table is played back by using every byte as data, or only intermittent bytes, as desired. The address resolution bits determine whether or not every byte is used during playback. If the resolution bits are 000, all wavetable data bytes are read; if the resolution bits are 001, every other byte is used; if they are 010, every fourth byte is used, and so on.
-----	---	--

Resolution bits Address bits

2	1	0	used
0	0	0	1-16
0	0	1	2-17
0	1	0	3-18
0	1	1	4-19
1	0	0	5-20
1	0	1	6-21
1	1	0	7-22
1	1	1	8-23

Sound input and output specifications

The 7-pin molex connector is used for sound input and output to and from the Apple Cortland. The electrical specifications for these inputs and outputs are listed below in Table 6-3.

Signal Name	Pin number	Max	Unit
A/D input	1	2.5	V p-p, full scale conversion
analog ground	2	-	-
analog output	3	5 - 5	V p-p
channel addr 0	4	1	LSTTL load
channel addr 1	5	1	LSTTL load
channel strobe*	6	1	LSTTL load
channel addr 2	7	1	LSTTL load

* Channel Strobe goes low when the channel address is valid.

Table 6-3. Sound input and output electrical specifications.

Figure 6-7 shows an example of a demultiplexer circuit that can be used to produce stereo (2-channel) sound using the output from the DOC available at the 7-pin molex connector J-25. A more complex circuit would result in 15 unique sound channels.

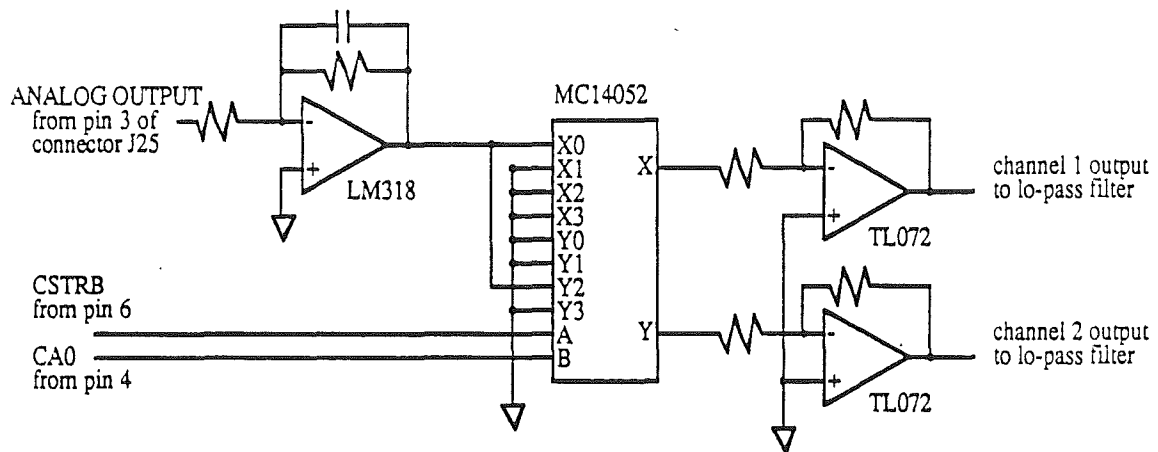


Figure 6-7. An example of a 2-channel demultiplexer circuit.

Sound Summary

The Apple Cortland provides new sound synthesis capabilities new to the Apple II family. The Sound Tools in the Cortland are there to use for manipulating the sound synthesizer. If you decide to bypass the tools, you must address the registers within the DOC and the Sound GLU, as well as the sound RAM. To do this you must

1. Set the Sound Control register bits for DOC or RAM access with the Address Pointer auto-increment option enabled.
2. Load the Address Pointers with the target address.
3. Read or write the data byte.

The addresses of the DOC registers for each oscillator are listed in Table 6-4.

DOC Register Addresses

Oscillator Number	Freq Register Low	Freq Register High	Volume Register	Data Register	Address Pointer	Control Register	Waveform Register
\$ 00	\$ 00	20	40	60	80	A0	C0
01	01	21	41	61	81	A1	C1
02	02	22	42	62	82	A2	C2
03	03	23	43	63	83	A3	C3
04	04	24	44	64	84	A4	C4
05	05	25	45	65	85	A5	C5
06	06	26	46	66	86	A6	C6
07	07	27	47	67	87	A7	C7
08	08	28	48	68	88	A8	C8
09	09	29	49	69	89	A9	09
0A	0A	2A	4A	6A	8A	AA	CA
0B	0B	2B	4B	6B	8B	AB	CB
0C	0C	2C	4C	6C	8C	AC	CC
0D	0D	2D	4D	6D	8D	AD	CD
0E	0E	2E	4E	6E	8E	AE	CE
0F	0F	2F	4F	6F	8F	AF	CF
10	10	30	50	70	90	B0	D0
11	11	31	51	71	91	B1	D1
12	12	32	52	72	92	B2	D2
13	13	33	53	73	93	B3	D3
14	14	34	54	74	94	B4	D4
15	15	35	55	75	95	B5	D5
16	16	36	56	76	96	B6	D6
17	17	37	57	77	97	B7	D7
18	18	38	58	78	98	B8	D8
19	19	39	59	79	99	B9	D9
1A	1A	3A	5A	7A	9A	BA	DA
1B	1B	3B	5B	7B	9B	BB	DB
1C	1C	3C	5C	7C	9C	BC	DC
1D	1D	3D	5D	7D	9D	BD	DD
1E	1E	3E	5E	7E	9E	BE	DE
1F	1F	3F	5F	7F	9F	BF	DF

Table 6-4. DOC register addresses.

INSERT ENSONIQ DATA SHEET HERE.

Cortland Hardware Reference Manual

Chapter 7

Apple DeskTop Bus

Introduction

The Apple DeskTop Bus (ADB) is a method for connecting input devices (such as a keyboard or a mouse) with the Apple Cortland computer. The ADB consists of a Keyboard Microcontroller chip and the Apple DeskTop Bus cabling. Figure 7-1 shows the relationship of the ADB components in the Apple Cortland computer.

The Keyboard Microcontroller controls devices on the bus by receiving commands from the 65C816 microprocessor, and then sending appropriate ADB commands to and receiving data from, the input devices on the bus. Microcontroller commands (those received from the 65C816) are located in ROM.

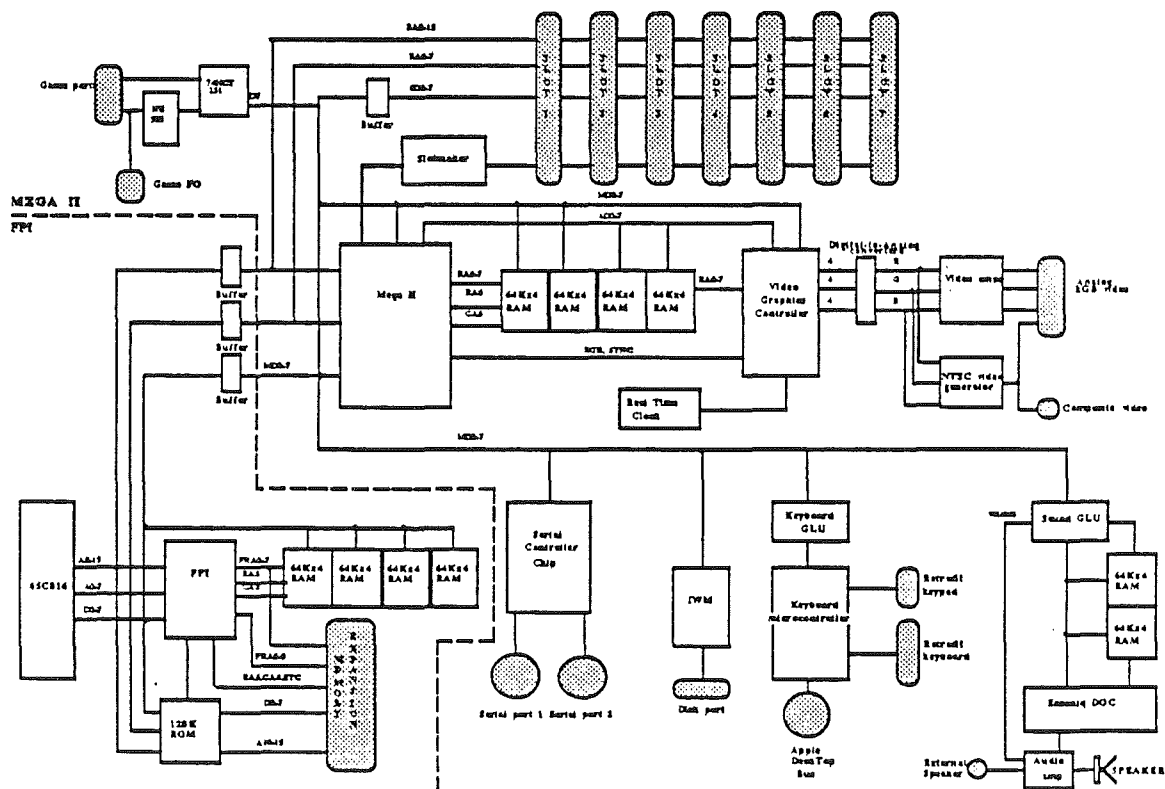


Figure 7-1. The ADB within the Apple Cortland.

To find more information regarding the ADB toolkit, refer to the *Cortland Firmware Reference* manual.

Note: To keep compatibility with future Apple // products using ADB, use the Apple DeskTop Bus Toolkit commands in ROM.

This chapter describes the physical and network layers of the (ADB) as it is used in the Apple Cortland computer. For the remainder of this chapter, the computer will be referred to as the host and the input devices (for example, a keyboard or a mouse) connected to the bus are referred to as devices.

ADB input devices are covered in *Chapter 9: Input devices*. For information specific to the keyboard and the mouse, refer to Chapter 9.

The input bus

All input devices share the input bus with the host. This bus consists of a 3-wire cable and uses 4-pin mini-DIN jacks at the host and at each device. Figure 7-2 shows the pin assignments of the connectors. ADB devices may use the +5 volt power supplied by the bus, but must not draw more than 500 mA total for all devices. All devices are connected in parallel, using the signal, power and ground wires. Cables should be no longer than 5 meters, and cable capacitance should not exceed 100 picofarads per meter.

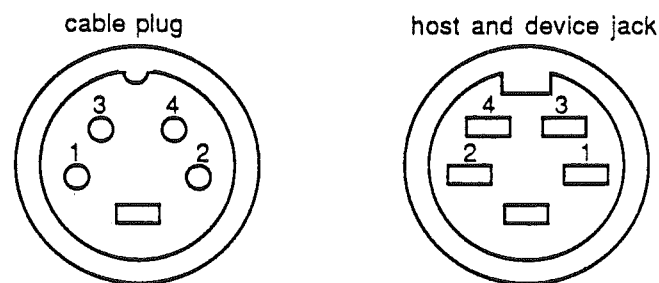


Figure 7-2. Mini DIN connector pin configuration used in the ADB.

The pin definitions are

Pin	Description
1	data
2	reserved
3	+5 power supply @ 500 mA for all devices
4	signal and power ground

The keyboard microcontroller

The keyboard microcontroller is an intelligent controller IC which oversees the Apple DeskTop Bus. The M50740 microcontroller (micro) uses a superset of the 6502 instruction set, and contains 96 bytes of RAM and 3K of ROM. The keyboard micro operates asynchronously, issuing commands on the bus and transmitting data to and receiving data from the bus devices. Use the ADB commands in the ROM toolbox to communicate with the ADB.

To find out how to use the Toolbox in the system ROM, see the *Cortland Firmware Reference* manual.

Keyboard GLU

The Keyboard General Logic Unit (GLU) works together with the keyboard microcontroller to form an intelligent input-device interface. The Keyboard GLU, located on the motherboard, uses two independent data buses that serve as a communications interface between the keyboard microcontroller and the system bus. This interface is accomplished by using multiple internal read/write registers to store keyboard data, key modifiers, mouse X/Y-coordinates, command data and status information.

Keyboard GLU Registers

The keyboard GLU contains seven data and control registers. These are used for storing keyboard data and commands, key modifiers, mouse X and Y coordinates, and status information. The registers are

- Mouse X-Coordinate register
- Mouse Y-Coordinate register
- Keyboard Status register
- System Status register
- Command register
- Keyboard Data register
- KeyMod register

All registers, except the status registers, have a status flag that is set to 1 when the register is written to, and cleared to 0 when the register is read. Each of the keyboard data, mouse, and data registers also have an interrupt flag that generates system interrupts, if interrupts are enabled. These status and interrupt flags are located in the status register. The registers are described in the following sections.

Keyboard GLU Status Register

The keyboard status register contains information on the status of each of the registers, as well as control of the keyboard strobe. Figure 7-3 shows the format of the keyboard GLU status register at location \$C0xx and is followed by description of each bit.

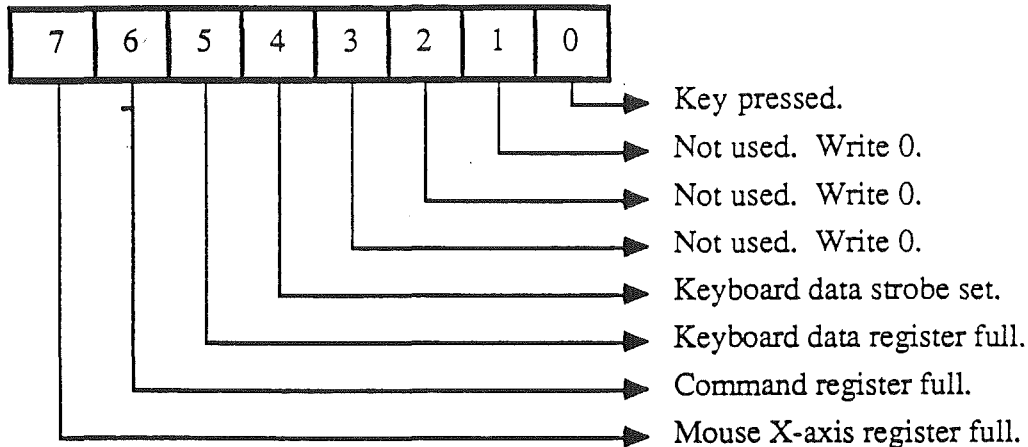


Figure 7-3. Keyboard GLU status register at \$C0xx.

Bit Value Description

7	1	When this bit is 1, the Mouse X-axis register full (read-only).
	0	When this bit is 0, the Mouse X-axis register is empty.
6	1	When this bit is 1, the Command register is full (read-only).
	0	When this bit is 0, the Command register is empty.
5	1	When this bit is 1, the Keyboard data register full.
	0	When this bit is 0, the Keyboard register is empty.
4	1	When this bit is 1, the Keyboard data strobe is set.
	0	When this bit is 0, the Strobe is cleared.
3-1	-	Not used; must be 0.
0	1	A key has been pressed.
	0	All keys are up.

You can set the Keyboard Strobe flag by writing to bit 7 in the Keyboard Data register at location \$C000, and clear it by writing to location \$C010.

The Mouse Register Full flag is set when the Keyboard micro writes to the mouse X-axis register. The flag is cleared when the Keyboard micro reads the system status register and reads the mouse register twice; the first mouse-register read returns the X-coordinate, and the second read returns the Y-coordinate.

The Data Register Full flag is set when the Keyboard micro writes to the data register; it is cleared when the system reads the system status register and the data register.

Keyboard/Mouse Status Register

The system status register, located at \$C027 contains flags that relate to mouse and keyboard data and status. Figure 7-4 shows the format of the Keyboard/Mouse Status register, followed by a description of each bit.

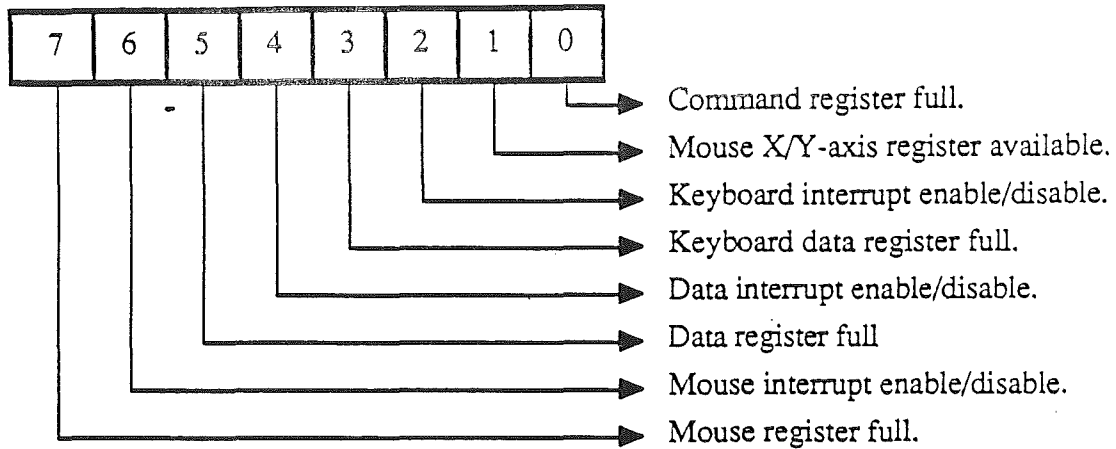


Figure 7-4. System Status register at \$C027.

Bit Value Description

Bit	Value	Description
7	1	When this bit is 1, the Mouse data register at \$C024 is full (read only bit)
	0	When this bit is 0, the Mouse register is empty.
6	1	When this bit is 1, the Mouse interrupt is enabled (read/write bit)
	0	When this bit is 0, the Mouse interrupt is disabled.
5	1	When this bit is 1, the Data register is full (read only bit)
	0	When this bit is 0, the Data register is empty.
4	1	When this bit is 1, the Data interrupt is enabled (read/write bit)
	0	When this bit is 0, the Data interrupt is disabled.
3	1	When this bit is 1, the Keyboard data register is full (read only bit)
	0	When this bit is 0, the Register is empty.
2	1	When this bit is 1, the Keyboard data interrupt is enabled (read/write bit)
	0	When this bit is 0, the Interrupt is disabled.
1	0	Mouse X-axis register available (read only bit)
	1	Mouse Y-axis register available.
0	1	Command register full (read only bit)
	0	Register empty.

The Command Full flag is set to 1 when the system writes to the command register; and cleared to 0 when the Keyboard micro reads the command register.

The Keyboard Data Full flag is set when the keyboard writes data into this register; it is cleared when the system reads the system status register and the keyboard data register.

Keyboard Data register

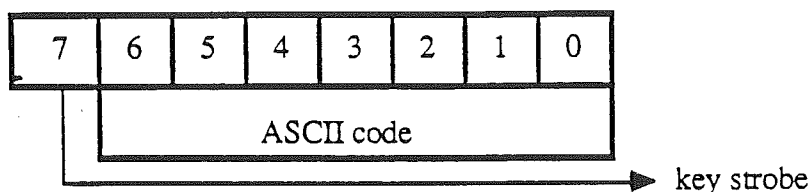


Figure 7-5. The Modifier Key register at \$C000.

Bit Value Description

- 7 - To read the keyboard data, set this bit by writing a 1 to bit 7. Reading bits 6 through 0 will result in valid ASCII data. This bit must be cleared after reading the data by writing to address \$C010.
- 6-0 - ASCII data from the keyboard.

Modifier Key register

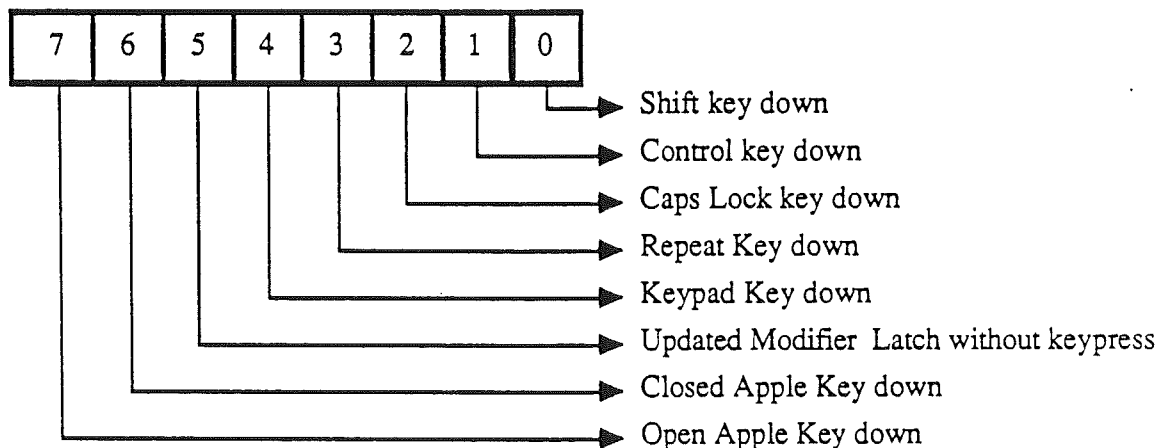


Figure 7-6. The Modifier Key register at \$C025.

Bit Value Description

- 7 1 When this bit is 1, the Open Apple key has been pressed.
 0 When this bit is 0, the Open Apple key has not been pressed.
- 6 1 When this bit is 1, the Closed Apple key has been pressed.
 0 When this bit is 0, the Closed Apple key has not been pressed.
- 5 1 When this bit is 1, the Modifier Key latch has been updated, but no key has
 0 - been pressed.
- 4 1 When this bit is 1, a keypad key has been pressed.
 0 When this bit is 0, a keypad key has not been pressed.

3	1	When this bit is 1, the Repeat function is active.
	0	When this bit is 0, the Repeat function is inactive.
2	1	When this bit is 1, the Caps Lock key has been pressed.
	0	-When this bit is 0, the Caps Lock key has not been pressed.
1	1	When this bit is 1, the Control key has been pressed.
	0	When this bit is 0, the Control key has not been pressed.
0	1	When this bit is 1, the Shift key has been pressed.
	0	When this bit is 0, the Shift key has not been pressed.

Mouse Data register

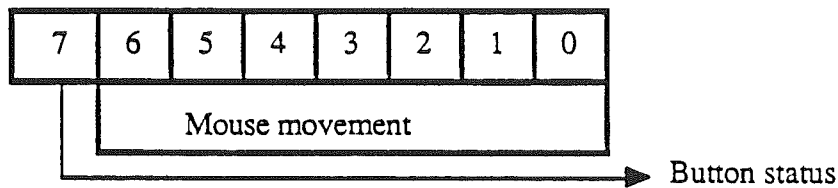


Figure 7-7. The Mouse Data register at \$C024.

Bit Value Description

7	1	Mouse button up.
	0	Mouse button down
6-0	-	The relative mouse movement data is returned here.

Command/Status register

The Command/Status register is a dual-function register used to communicate to the ADB. To send a command to a device on the bus, write the command byte to this register at address \$C026. To check the status of the specific device, read this register at the same address. Figure 7-8 shows the format of the Command/Status register when it is read.

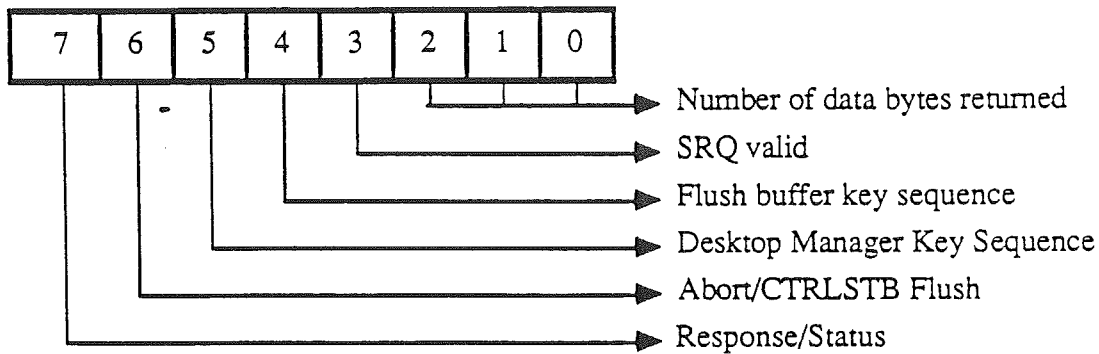


Figure 7-8 The Command/Status register at \$C026.

Bit Value Description

7	1	When this bit is 1, the keyboard micro has received a response from an ADB device previously addressed.
	0	No response available.
6	1	When this bit is 1, and only this bit in the register is 1, the keyboard micro has encountered an error and has reset itself. When this bit is 1 and bit 4 is also 1, this indicates that the keyboard micro should clear the key strobe (bit 7 in the data register at \$C000).
	0	-
5	1	When this bit is 1, the Apple, Control, and Reset keys have been pressed simultaneously. This condition is usually used to initiate a cold boot.
	0	Reset key sequence has not been pressed.
4	1	When this bit is 1, the Apple, Control, and Delete keys have been pressed simultaneously. This condition will result in the keyboard micro flushing all internally buffered commands.
	0	Buffer flush key sequence has not been pressed.
3	1	When this bit is 1, a valid Service Request is pending. The keyboard micro will then poll the ADB devices and determine which has initiated the request.
	0	No Service Request pending.
2-0	-	The number of data bytes to be returned from the device is listed here.

Bus Communication

The host carries communication on the bus by sending to a device either commands or data. A device can respond to commands by sending data to the host. This form of communication uses strings of bits, each making up a packet. A data transfer or transaction consists of a complete communication between the host and a device; for example, it may be a command packet sent by the host to request data from a device, followed by a data packet sent from the device to the host.

Figure 7-9 shows how duty-cycle modulation represents bits on the bus. A low period of less than 50% of the bit-cell time is interpreted as a 1. A low period of greater than 50% of the bit-cell time is interpreted as a 0.

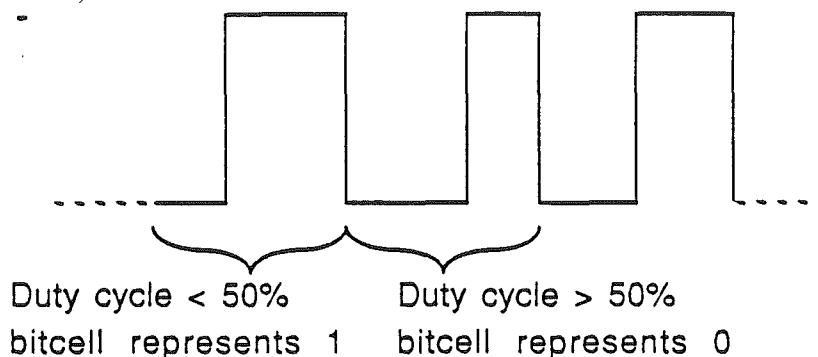


Figure 7-9. Bit representation via duty cycle modulation.

Signals

Certain transactions are neither commands nor data transactions. These are special transactions that the host uses to broadcast status globally to all devices on the bus. There are four special transactions in this group: **Attention**, **Sync**, **Reset**, and **Service Request**.

Attention and Sync

The start of every command is signaled by a long, low, attention pulse that the host sends on the bus. This is followed by a short, high, sync pulse that signals the beginning of the initial bus timing. The falling edge of the sync pulse is used as a timing reference after which the first command bit follows. Figure 7-10 shows the format of the Attention and Sync signals.

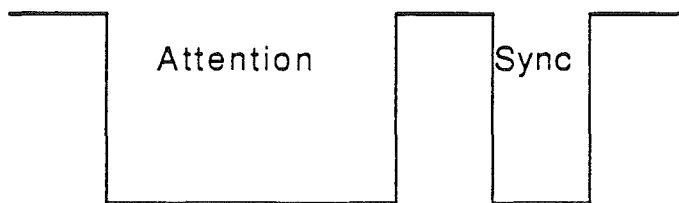


Figure 7-10. Attention and Sync pulses.

Reset

Reset issues a break on the bus. Only the host may issue this signal which signals all devices to reset. By holding the bus low for a minimum of 2.8 milliseconds, a reset is initiated.

Service Request

Service request is a transaction that devices can use to signal the host that they require service, such as when there is data to send to the host. Only a device can issue a service request. Following any command packet, a requesting device can signal a service request by holding the bus low during the low portion of the stop bit of the command transaction. This lengthens the stop by a minimum of 140 milliseconds beyond its normal bitcell boundary. This lengthened stop bit indicates to the host that a service request is desired. Figure 7-11 shows the format of the Service Request signal.

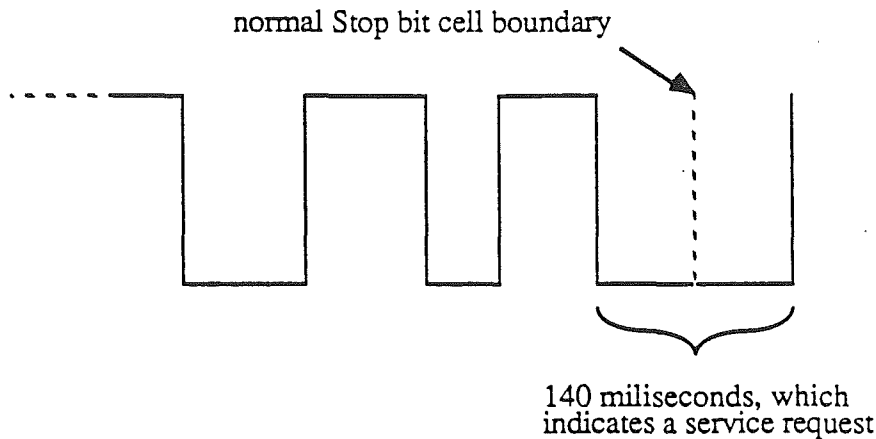


Figure 7-11. Service Request

A device will signal Request Service repeatedly until it is served. When a device has requested service (at this point the host does not know *which* device sent the request), the host will poll each of the devices by sending a **Talk register 2** command.

When the host commands the requesting device to **Talk**, the device is considered served and does not send a **Request Service** signal again until it needs to be served again. The host can enable and disable the ability for a device to send **Request Service** at any time. ADB keyboards are prohibited by the Apple Cortland from issuing Service Requests. All other ADB devices may issue Service Requests.

Reset

The host can reset all devices on the bus by holding the bus low for a minimum of 250 milliseconds. Upon detecting a reset on the bus, devices will reset and place themselves in listen mode.

Transactions

A command (Talk, Listen, Flush) initiates a transaction. The sequence of the command is as follows:

- 1) an Attention signal
- 2) a Sync signal
- 3) eight Command bits

4) one Stop bit

To synchronize the end of the transaction, the command transmits the stop bit after the last command bit-cell. Then the transaction is complete and the host releases its control of the bus (the bus is always floating in a high state until a device or the host initiate a transaction).

Apple DeskTop Bus Peripherals

Each device on the bus has an address. There is only one active talker on the bus at a time; this may be the host or an addressed device. A device addressed to talk (with data to send) releases control of the bus after it sends the data. If a device has been addresses but has no data to send, it releases control of the bus immediately and allows the host to time out (waiting for data, none arrives). The host may also send data to the addressed device in a separate packet, after it sends a listen command to the device.

Each peripheral device has a four-bit command address that identifies its device-type. A total of 16 addresses are available, which means that a maximum of 16 unique devices may be on the bus. A device always responds to its address when there is a power-on or a reset signal.

Commands

Only the host can send commands. There are two types of commands: the **Talk** command is used for data transaction from a device to the host; the **Listen** command is used for a data transaction from the host to a device.

A command is an eight-bit word that has a specific syntax:

- 1) a four-bit field that specifies the address of the desired device. The addresses ranges from 0-15 (A3-A0)
- 2) a four-bit command and register address code.

Bit No:		Command
7 6 5 4	3 2 1 0	
XXXX	0000	Send Reset*
A3-A0	0001	Flush
XXXX	0010	Reserved
XXXX	0011	Reserved
XXXX	01XX	Reserved
A3-A0	10RR	Listen
A3-A0	11RR	Talk

X = ignored

R = register number

* Forces RESET signal on FDB

Table 7-1. Command Byte Syntax

To allow for future expansion of the command structure, the Apple Cortland has reserved a group of instructions which are currently treated as no-ops.

Talk

When the host addresses and requests a device to Talk, the device must respond with data before the host times out (does not receive data within the specified time). The selected device performs its data transaction and releases the bus.

Listen

When the host addresses and requests a device to Listen, it is enabled to receive the data bits that the host places on the bus. The host performs its data transaction. After the stop bit that follows the data is received, the transaction is complete and the device releases the bus. If a listening device detects another command on the bus before it receives any data, the original transaction is immediately considered complete and the device releases the bus.

Registers

All devices have four locations to receive data. These are :

- Register 0, Talk: Data register, device specific.
- Register 0, Listen: Data register, device specific.
- Register 1, Talk: Data register, device specific.
- Register 1, Listen: Data register, device specific.
- Register 2, Talk: Data register, device specific.
- Register 2, Listen: Soft addressed devices: Device specific.
- Register 3, Talk: Status information, that is, device address handler.
- Register 3, Listen: Status information, that is, device address handler.

Collision Detection

All devices must be able to detect collisions. If a device is attempting to output a bit and the data line is forced (either high or low) by another device, it has lost a bit in collision with the other device. If another device sends data before the device is able to assert its start bit, it has lost a collision. The losing device should immediately *untalk* itself and preserve the data that was being sent for retransmission. The device sets an internal flag if it loses a collision.

Note: Devices using internal clocks that operate within $\pm 1\%$ should attempt to assert their start bit at a random time within the limits of the line turn-around time .

Error Conditions

If the bus hangs low, all devices reset themselves and output a one. If a command transaction is incomplete by staying high beyond the maximum bit-cell time, all devices ignore the command and listen for another attention signal.

Network Layer (ADB Types)

The Network Layer accommodates Normal Devices and Extended Address Devices.

Normal Devices

A normal device optionally has a device, called the *activator*, on it to indicate activity. The activator can be a special key on a keyboard or a mouse button.

To aid in collision detection, the address portion of the address field of Register 3 is replaced with a random number in response to a Talk R3 command. Normal devices will change their Register 3 to the data received when they receive a Listen R3 command, no collision detected, and activator inactive is true.

At the systems level, a host can change the address of normal devices by forcing the collision of devices sharing the same address. By issuing a Talk R3 command and following it with a Listen R3 command, with a new address in bits 8 to 11 of the data, all devices that detect collisions are moved to the new address. This process can be repeated at new addresses until the response to the Talk R3 command is a time-out. This can be used to identify and relocate multiple devices of the same type after initialization of the system.

At the applications level, addresses can be changed by displaying a message requesting a user to use the activator. The host then issues a Listen R3 command to a new address and all devices, except the one with the activator being used, are moved. This method can be used to identify and locate individual devices in multi-user applications. Figure 8-12 shows the format of this register.

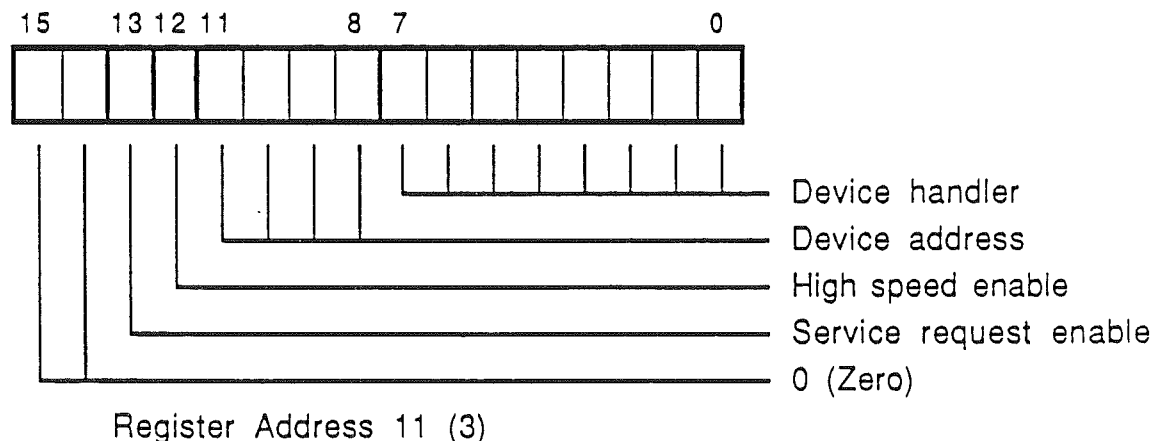


Figure 7-12. Register 3.

Extended Address Devices

Extended-address devices have the same command address and a unique 16-bit extended address that is stored in the device. Their command address cannot be changed. On power-up or RESET, they only accept the Listen-R2 command in which the data match their stored address. When enabled, they respond to all commands addressed to them. These devices become disabled after receiving a Listen R2 command in which the data do not match their stored address.

Register 3

The function of a device and the use of its data by the host are controlled by a handler that is stored by the device in register 3. The host changes the handler with a Listen-R3 command. If the receiving device is able to function with the new handler, it is stored and sent in response to a Talk-R3 command. Figure 7-13 shows the format of register R3 as used in a keyboard. Figure 7-15 shows the format of register R3 as used in a mouse.

Handler \$FF is reserved for the self test mode for all devices. Handler \$00 in response to a Talk is reserved to indicate a failed self test. Handler \$00 sent with a Listen is reserved to indicate that the device is only to change the address portion of R3. The following are examples (tables 23 and 24) of how the handlers for keyboards (coded device) and mice (relative device) could interpret data received from a Talk R0 command.

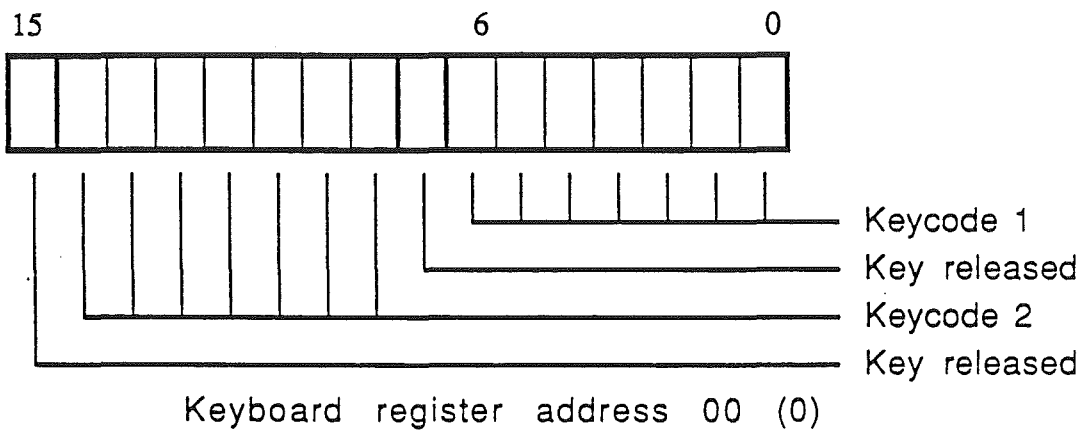


Figure 7-13. Keyboard Register

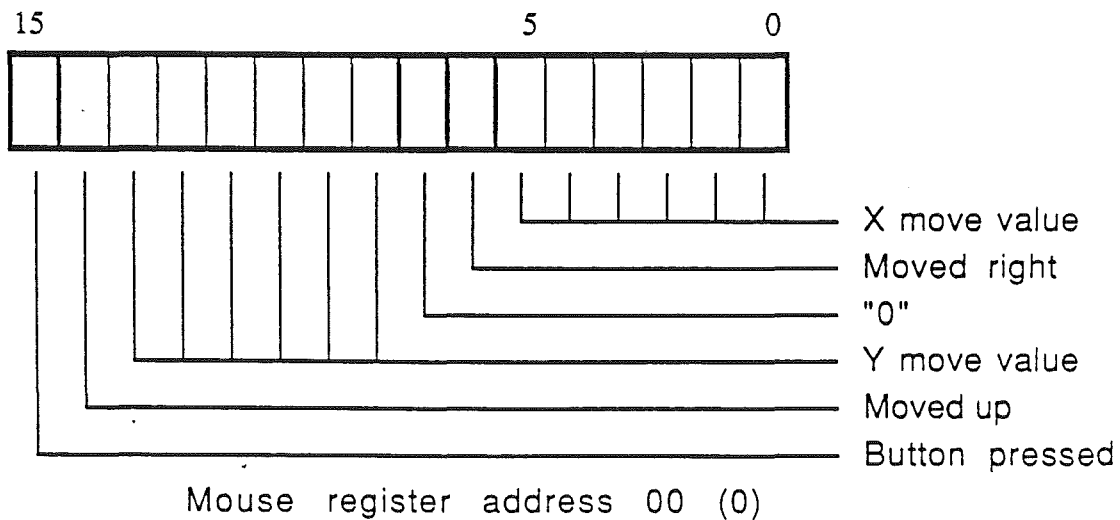


Figure 7-14. Mouse Register

Service Request

The Listen-R3 command is also used to enable and disable **Service Request**. Service Request is enabled on the bus by setting the R3 enable bit to 1; it is disabled by setting the bit to 0. This is useful in systems where the Service Request response time in a polled system is longer than desired. When only specific devices are required for an application, the others can be disabled.

Cortland Hardware Reference Manual

Chapter 8

The disk port

Introduction

The Apple Cortland computer can use either 5.25-inch 140K disk drives or the newer 3.5-inch 800K disk drives. The disk port connector at the rear panel is compatible with both types of Apple disk drives. This chapter describes the disk port connector with the Cortland. Figure 8-1 shows the Apple Cortland block diagram and position of the disk port within the Apple Cortland configuration.

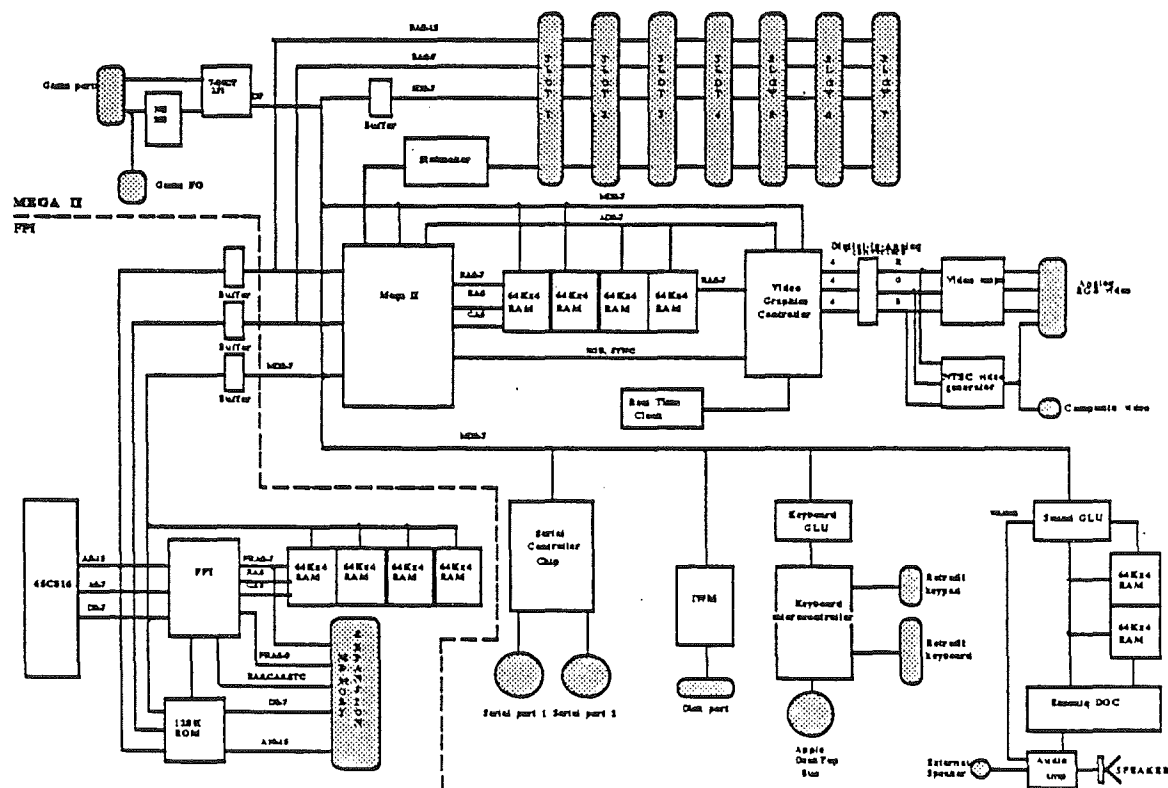


Figure 8-1. Cortland block diagram.

Apple II Compatibility

The Apple Cortland uses the same disk drive interface as the Apple //c and //e. Programs written for both of these earlier computers will run on the Cortland. The Cortland firmware recognizes ProDOS block device calls and SmartPort interface calls to both the Apple UniDisk 3.5 micro-floppy and Apple DuoDisk 5.25-inch mini-disk drives.

Sidebar:

To find out how to use the ProDos block device calls see the *ProDos Reference* manual. To find out how to use the SmartPort interface calls, see the *Cortland Firmware Reference* manual.

The disk port connector

The disk port connector is located at the rear of the Cortland case. It is a DB-19 connector which contains 19 pins. Figure 8-2 shows the connector and Table 8-1 lists the signals and their descriptions.

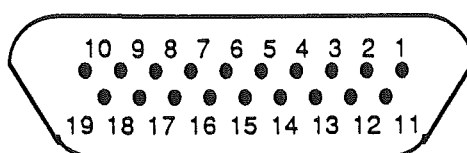


Figure 8-2. The disk port connector.

1,2,3	GND	Ground reference and supply
4	3.5DISK	3.5- or 5.25-inch drive select
5	-12V	-12 volt supply
6	+5V	+5 volt supply
7,8	+12V	+12 volt supply
9	DR2	Drive 2 select
10	WRPROTECT	Write protect input
11	PH0	Motor phase 0 output
12	PH1	Motor phase 1 output
13	PH2	Motor phase 2 output
14	PH3	Motor phase 3 output
15	WREQ	Write request
16	HDSEL	Head select
17	DR1	Drive 1 select
18	RDDATA	Read data input
19	WDATA	Write data output

Table 8-1. The disk port signals.

Warning: The power connections on this disk port are for use by the disk drive only. Do not use these connections for any other purpose. Any other use of these supplies may damage the voltage regulator within the computer.

The IWM

The disk port interface is enhanced by the Integrated Woz Machine (IWM) which simplifies the microprocessor's task of reading and writing serial data to and from each floppy disk drive. To perform disk operations, the microprocessor simply reads or writes control and data bytes to or from the six IWM registers.

The disk port is mapped as an internal device at addresses \$C0E0 through \$C0EF. These are the same addresses as in the Apple //c and //e. Table 8-2 shows these locations and their function.

\$C0E0	stepper motor phase 0 low
\$C0E1	stepper motor phase 0 high
\$C0E2	stepper motor phase 1 low
\$C0E3	stepper motor phase 1 high
\$C0E4	stepper motor phase 2 low
\$C0E5	stepper motor phase 2 high
\$C0E6	stepper motor phase 3 low
\$C0E7	stepper motor phase 3 high
\$C0E8	spindle motor enabled
\$C0E9	spindle motor disabled
\$C0EA	drive 0 select
\$C0EB	drive 1 select
\$C0EC	Q6 select bit low
\$C0ED	Q6 select bit high
\$C0EE	Q7 select bit low
\$C0EF	Q7 select bit high

Table 8-2. Disk port soft switches.

Soft switches Q6 and Q7 are select bits for accessing registers within the IWM. By setting or clearing Q6, Q7 and Spindle Motor bits, you may read or write to one of the registers.

Disk Interface register

The Disk interface register (\$C031) serves as a control register for the disk drive. By writing to this register, the programmer selects the type of disk drive being used and which side of the diskette to access.

This register uses only two bits which are both cleared on reset. When the Disk register is read, zeros are returned in the unused positions (bits 5-0). Figure 8-3 shows the format for this register. Descriptions of each bit are also listed below.

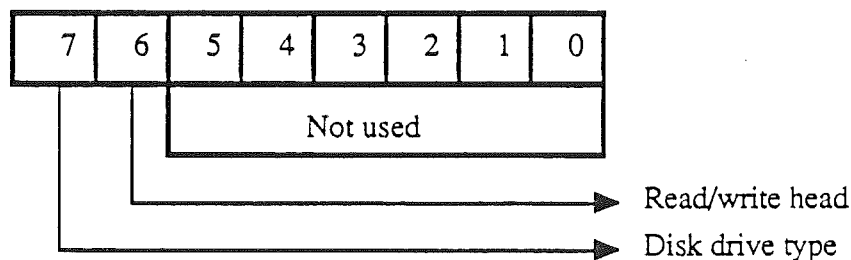


Figure 8-3. Disk Interface register at \$C031.

Bit	Value	Description
7	1	Read/write head select bit. A 1 in this position selects head 1.
	0	A 0 selects head 0.
6	1	Disk drive select bit. A 1 in this position selects 3.5-inch disks.
	0	A 0 selects 5.25-inch disks.
5-0	-	Not used. Set to 0.

Chapter 9

The memory expansion slot

Introduction

The extended memory card slot allows you to add a memory card holding up to eight Megabytes of RAM and 896K of ROM memory. It supports additional memory only and is not to be used for any other purpose. One megabyte or 4 megabyte RAM cards can be constructed by using 256K x 1 or 1M x 1 RAM ICs. Figure 9-1 shows a block diagram of the Cortland.

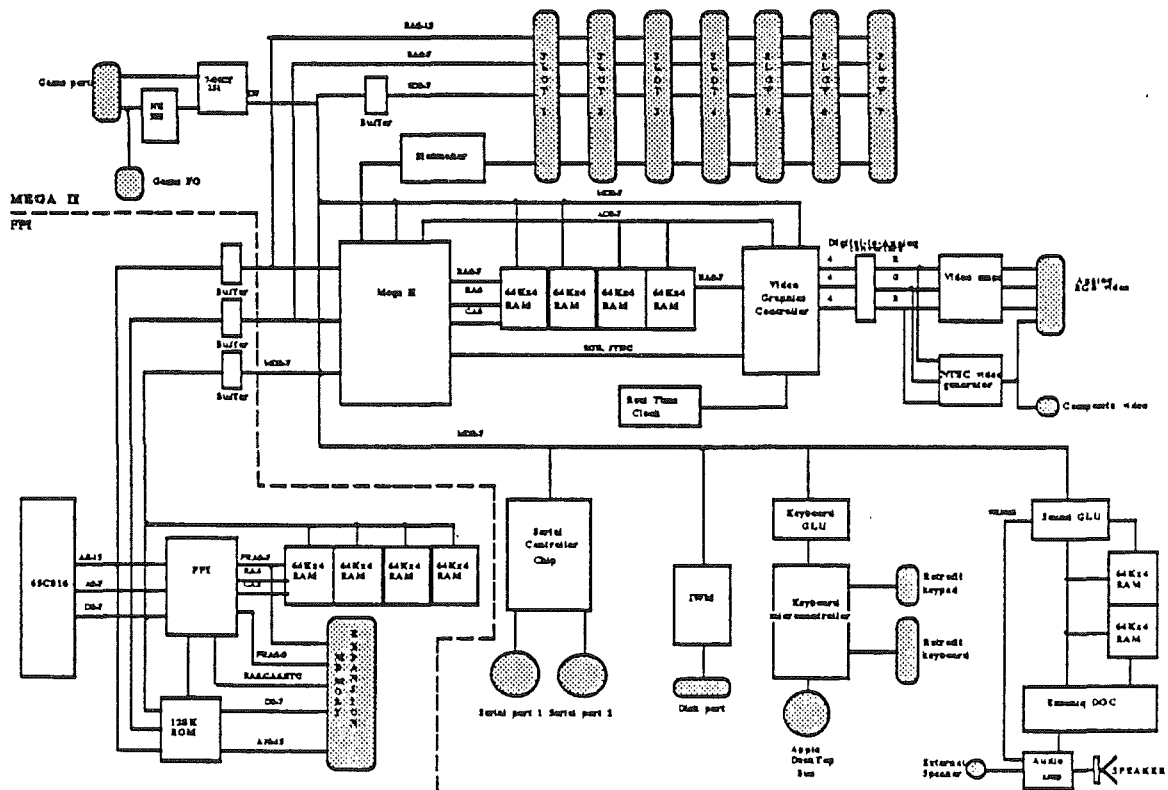


Figure 9-1. Cortland block diagram.

Extended RAM

Up to 4 megabytes (64 banks of 64 K-bytes each) of RAM can be designed in the extended memory card. This would be organized as four rows (eight chips per row) of RAMs with

each row holding either 256K bytes or 1M byte. This requires the use of 256 K-bit x 4 bit (resulting in 1 megabyte total) or 1 megabit x 1 bit RAMs (yielding 4 megabytes total).

To control and select individual rows of RAM, the FPI provides /CRAS, /CCAS, CROW0, and CROW1 signals. Signals /CRAS (Card Row Address Strobe) and /CCAS (Card Column Address Strobe) are the basic memory timing signals common to most dynamic RAMs. Signals CROW0 (Card ROW select 0) and CROW1 (Card ROW select 1) are row selects which, when taken as a pair, indicate the row number to be accessed. Typically, CROW0 and CROW1 are used as the select signals for a dual 1-of-4 decoder (74F139 or equivalent) that demultiplexes /CRAS and /CCAS into a separate /RAS and /CAS for each 8-chip segment.

Extended RAM Mapping

Figure 9-2 depicts a 1-megabyte extended RAM card using four rows of 256 K-bytes per row totalling 1 megabyte. The RAM banks above bank \$11 are *ghosts* (repeat images) of the RAM in banks \$2-11. A partially populated card causes holes in the memory map unless there is an option on the card to alter the address decoding. Therefore, contiguous memory for banks \$2 thru 11 is available only for 256 K-, 512 K- and 1 mega-byte expansion cards.

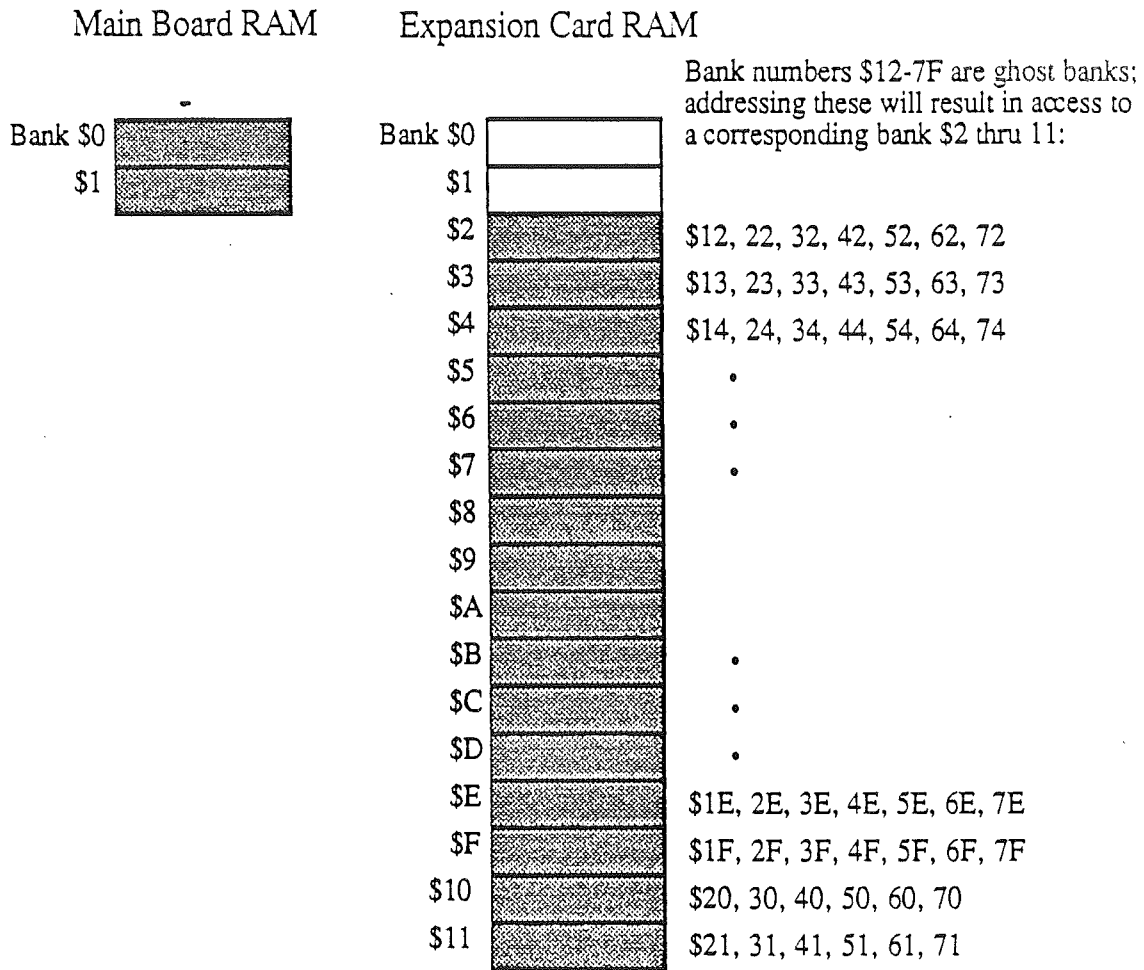


Figure 9-2. Extended RAM Mapping

MSIZE

A new input to the memory expansion slot, MSIZE, flags the type of memory chips being used on the memory expansion card.

If the MSIZE pin is tied to ground (when using 256 K-bit RAMs), the FPI multiplexes 18 address bits onto RA0-8 and generates the CROW0-1 row selects for rows of 256 K-bytes. If the MSIZE pin is not connected (for 1 M-bit RAMs), the FPI multiplexes 20 address bits onto RA0-9 and generates the CROW0-1 row selects for rows of 1 M-bytes.

Ghosting

The RAM expansion card is enabled for accesses in banks \$2-\$80, but only provides 1 megabyte of actual RAM (banks \$2-11). Four eight-IC rows of RAMs on the card are individually selected by CROW0 and CROW1. For a 1 M-byte card with 256 K-byte rows (MSIZE=0), the selected RAM row number is given by the bank number mod 4. For banks \$0-\$1 (main board RAM), the extended memory card is not accessed. This method of card and row selection causes multiple images or *ghosting* of the RAM areas on the card; whenever locations above \$FFFFFF are addressed, locations in a corresponding low bank (\$2-11) are accessed.

Extended ROM

Additional ROM space up to 896 K-bytes is available in banks \$F0 to \$FD. To accomplish this, an additional bank-address latch-decoder is required on the memory card. The FPI provides a signal (CROMSEL.L) that selects one bank; however, the card must provide the additional decoding to select individual ROMs within the selected bank.

The extended memory card connector provides a group of signals to support dynamic RAM and additional general purpose signals to support ROM decoding and selection. Table 9-1 lists these signals.

Signal	Description
FRA0-FRA9	10 bits of multiplexed RAM address for RAM cycles -- least significant 10 bits of ROM address.
CROW0,1	2 bits select 1 of 4 RAM rows.
CRAS.L	RAM /RAS strobe.
CCAS.L	RAM /RAS strobe.
FR/W	Write enable to RAMs. R/W from CPU or DMA.
D0-D7	8 bits of bidirectional data -- CPU data bus.
CDIR.L	Card data buffer direction control. High to read card data.
CROMSEL.L	Card ROM select. Low for accesses to banks \$F0-FD.
PH2CLK	CPU clock. Rising edge indicates valid bank address on D0-D7.
MSIZE	Output from card. Indicates RAM row size.
A10-A15	The 6 high-order address bits. Used with ROMs.
14M	14 MHz clock signal.
VCC	+5v +/-5%. 600 ma absolute maximum.

Table 9-1. Memory Card Interface Signals

To control and select individual rows of RAM located on the ROM card, the FPI provides the /CRAS, /CCAS, CROW0, and CROW1 signals. The /CRAS and /CCAS signals are for basic memory timing common to most dynamic RAMs. The CROW0 and CROW1 are row select signals which, when taken as a pair, indicate the row number to be accessed. Typically, CROW0 and CROW1 are select signals for a dual 1-of-4 decoder (74F139 or equivalent) that demultiplexes /CRAS and /CCAS into a separate /RAS and /CAS for each row. Figure 9-3 shows a typical circuit for RAM row selection.

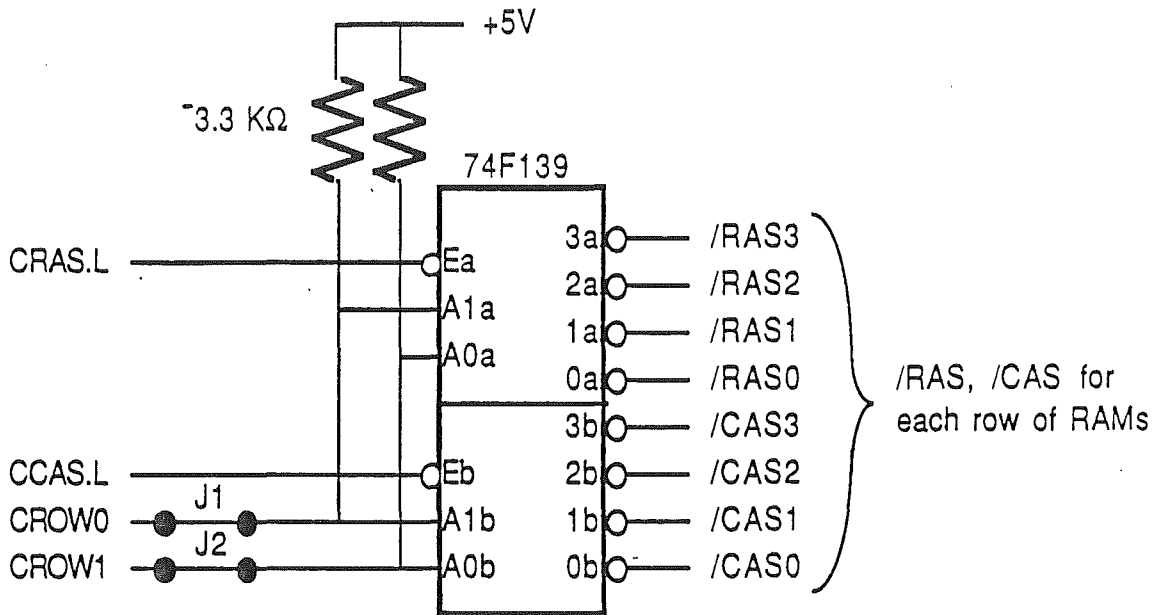
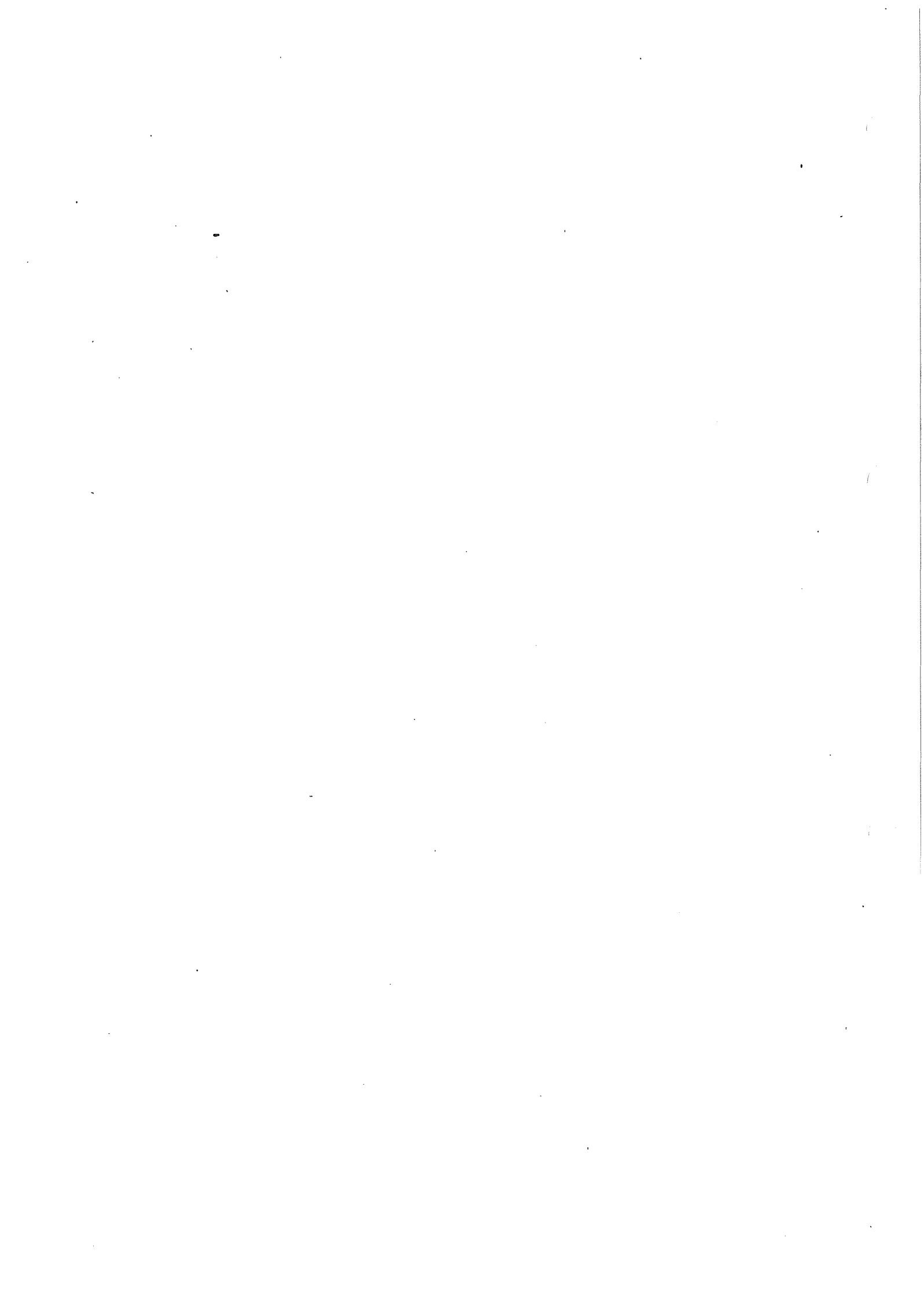


Figure 9-3. Example circuit for decoding the Extended Memory Card RAS/CAS signals.

Address Multiplexing

The FPI multiplexes the RAM addresses onto either eight, nine, or ten RAM address lines to provide support for RAM with 64K-, 256K- or 1M-bit RAM ICs. The main-board RAMs (banks \$0 and \$1) are 64 K-bit chips requiring eight address lines. The RAM expansion card can support 256 x 1-, 256 x 4-, 1M x 1-, or 1M x 4-bit RAMs. The expansion card manufacturer indicates word size of the RAMs on the memory card by the MSIZE signal from the card. (See MSIZE, above).



Chapter 10

Power Supply

Introduction

The Cortland power supply has the same four-supply, switching, load-sensing design as the Apple II, II+ and IIe models used. The following sections describe the design of this unit.

Function

The power supply changes high-voltage, alternating current (AC) into low-voltage, direct current (DC). The Cortland does this by using a switching-type power supply which allows a simple, maintenance-free operation.

Warning: The power supply contains dangerous voltages, and should be opened only by an authorized Apple service technician.

This power supply also contains special load-sensing circuitry; whenever it detects a short or a no-load condition, the supply will no longer provide voltages to the computer. This condition is easily recognized: the supply will emit two audible chirps per second. This condition will persist until you correct the situation or turn the power supply off.

Specifications

The Cortland power supply operates on regular household 120-volt Alternating Current. The supply provides +12 volts, -12 volts, +5 volts, -5 volts, and two ground return lines.

The power input requirements are 107 to 132V AC. The power output specifications are as follows:

- +12 volts at 0.5A
- -12 volts at 0.25A
- +5 volts at 2.5A
- -5 volts at 0.25A

The connector is a six-pin, female, keyed molex-type. Figure 10-1 shows its pin-out:

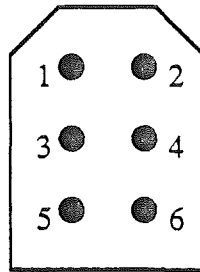


Figure 10-1. Power supply connector.

The pin assignments are

pin	description
1	ground
2	ground
3	+5 volts
4	+12 volts
5	-5 volts
6	-12 volts

Chapter 11

65C816 Microprocessor

Introduction

The microprocessor is the intelligence of the computer system. It is this device that recognizes the instructions encoded by the programmer and manipulates the other devices in the system (VGC, the Mega II, the DOC) which result in output such as video and sound. Figure 11-1 shows the Cortland block diagram and the relationship of the microprocessor to the rest of the computer.

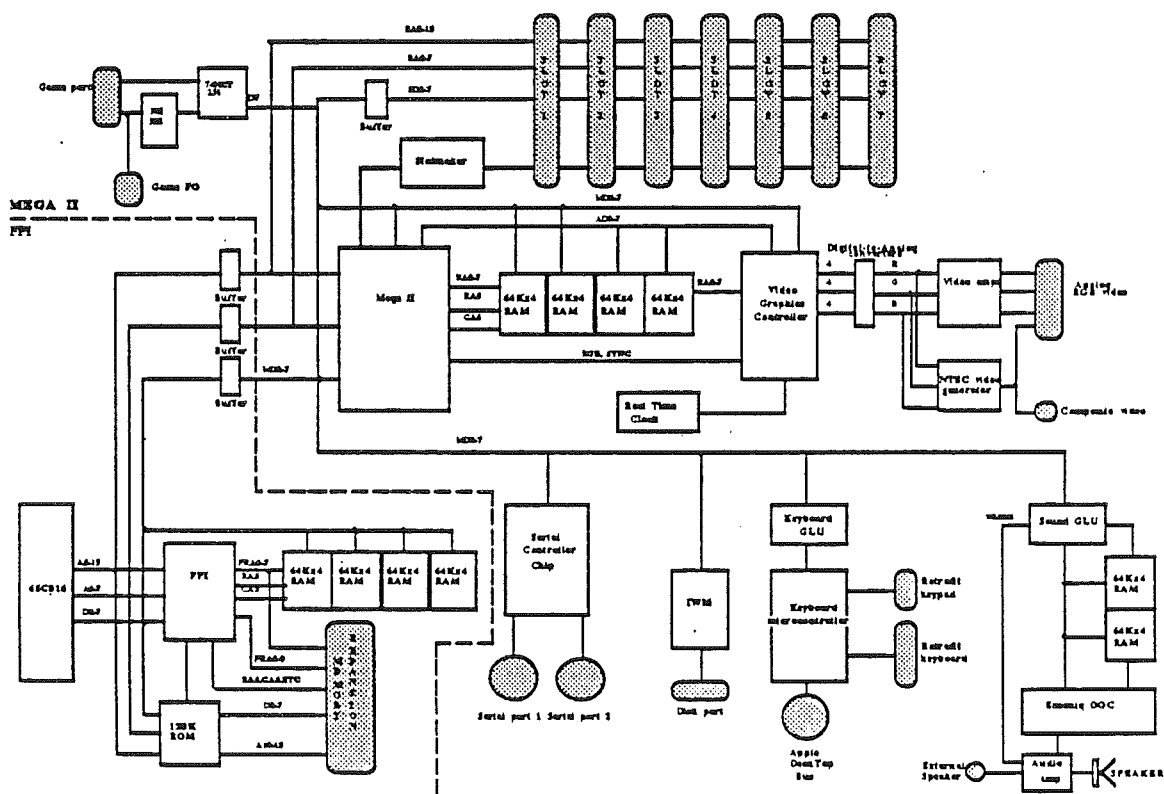


Figure 11-1. Cortland block diagram and the 65C816.

The Apple Cortland uses the 16-bit 65C816 microprocessor, a CMOS design based on the 6502 chip. The microprocessor provides this computer with greater computing power in these ways:

- 8 megabyte address range increases potential program and data size
- 16-bit internal data registers increases data-handling capability

- 2.5 MHz processor speeds computations

This chapter describes the new features of this microprocessor and its capability to emulate the 6502. Also, each of the 65C816 internal registers is described briefly.

CMOS is an abbreviation for *Complementary Metal Oxide Silicon*, which is one of several methods of semiconductor integrated-circuit fabrication. CMOS devices are characterized by their low power consumption.

65C816 Features

The new 65C816 microprocessor shares many characteristics with the 6502 and 65C02 as used in other Apple II-family computers. It also introduces new features not found in other Apple II computers. These are

- 16-bit accumulator
- 16-bit X and Y index registers
- Relocatable zero page
- Relocatable stack
- 24-bit internal address bus
- 8-bit data address bank register
- 8-bit program address bank register
- 11 new addressing modes
- 36 new instructions, for a total of 91 (all 256 op codes)
- Fast block-move instructions
- Ability to emulate 6502 and 65C02 8-bit microprocessors

For detailed descriptions of these features, refer to the manufacturer's data sheet at the end of this chapter. To learn how to implement these features, refer to the *Cortland Assembler Reference* manual.

The 65C816 microprocessor shares some features with the 6502 and 65C02 microprocessors used in previous Apple II models. Table 11-1 lists some of these features.

Characteristic	Year available	6502 1975	65C02 1983	65816 1985
Construction		NMOS	NMOS	CMOS
ALU bits		8	8	16
Address bus bits		16	16	24†
Data bus bits		8	8	8
Maximum memory		64K	64K	16M
Largest stack		256	256	64K
Defined opcodes		151	178	256
Addressing modes		13	15	24
Relocatable direct (zero) page?		No	No	Yes
6502 Software compatible?		Yes	Yes*	Yes
Fast block move instructions?		No	No	Yes
*sort of				
† high 8 bits multiplexed onto data bus				

Table 11-1. Some 6500-family ties.

NMOS is an abbreviation for *N-doped Metal Oxide Silicon*, which is one of several methods of semiconductor integrated-circuit fabrication.

The 65C816 is software compatible with the 6502 family of microprocessors, including the 6502 and the 65C02. Actually, the 65C816 has an emulation mode, in which it becomes an 8-bit 65C02. By emulating the 65C02, the 65C816 can execute most programs written for an Apple II computer.

The sixteen-bit 65C816

Sixteen-bit processor

In the Cortland, the 65C816 normally operates in either of two modes: 6502 emulation mode and 65C816 native mode. Figure 11-2 shows the sizes of the registers in emulation mode and in native mode. In emulation mode, the accumulator and index registers are 8 bits wide, and existing Apple II programs run the same as they do on any other Apple II model. In native mode, the accumulator and index registers are sixteen bits wide. The 65C816 also has several new and more powerful addressing modes that take advantage of its 24-bit addressing. The new addressing modes operate in either native mode or emulation mode, although the shorter registers in emulation mode make some of them ineffective.

Note: Native mode can also work with 8-bit data registers, with an additional accumulator, the B register. Apple does not recommend 8-bit native mode, but some internal routines use it, and developers are free to use it if they choose.

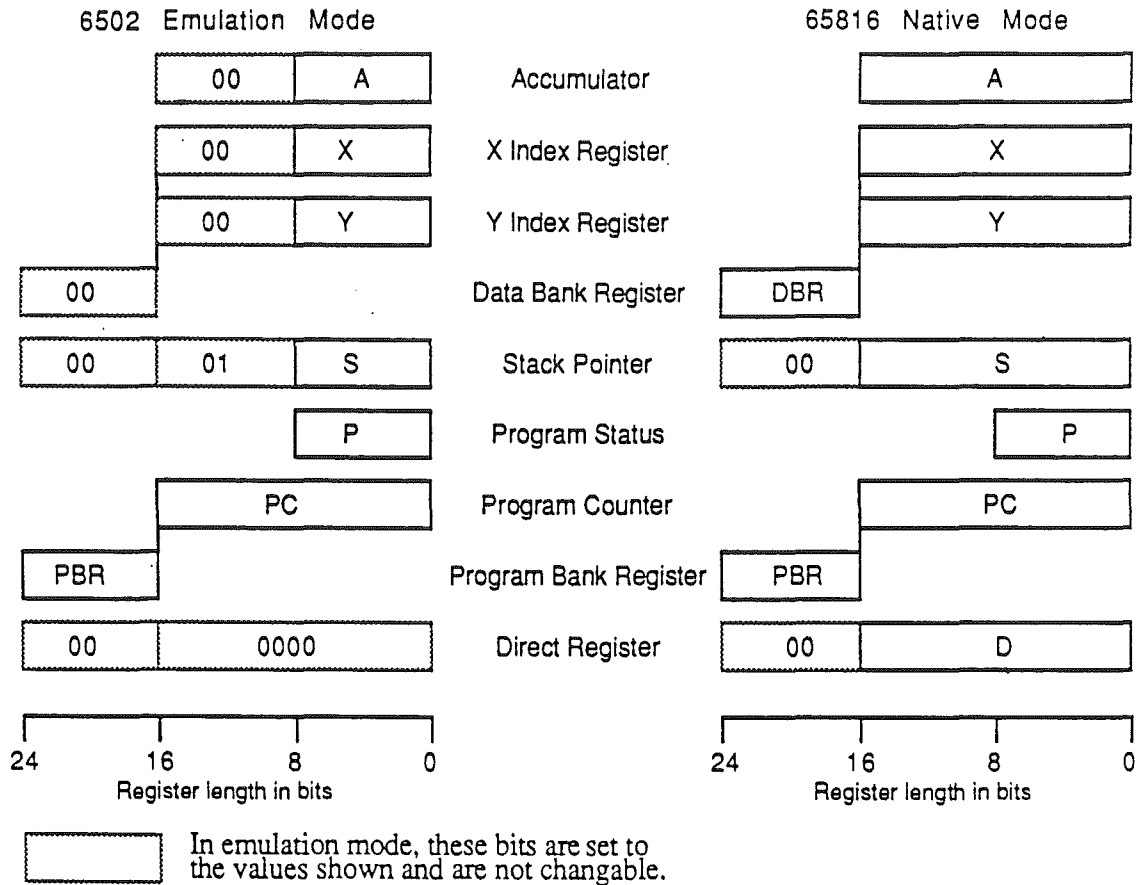


Figure 11-2. 65C816 registers.

Microprocessor differences

The 65C816 microprocessor differs from the 65C02 in several ways. This section describes some of those differences and their impact on program execution.

The registers

The 65C816 contains all of the registers found in the 65C02. In addition, the new microprocessor has three additional registers which make it a more powerful chip. These new registers provide additional addressing capability and greater data-handling capability. The nine registers within the 65C816 are described below.

To learn how to use the registers in the 65C816, see the *Cortland Assembler Reference* manual.

Accumulator

The Accumulator (also known as the Arithmetic Logic Unit—ALU) is a 16-bit register that holds all values while arithmetic and logical calculations are performed. The result of a calculation within this register effects the status bits within the Program Status register. In emulation mode, the upper eight bits are filled with zeroes that cannot be altered.

X Index register

The X Index register is a 16-bit register that is used as an address offset value when calculating an effective address. When the x bit is set, the upper eight bits are filled with zeroes that cannot be altered.

Y Index register

The Y Index register is a 16-bit register that is used as an address offset value when calculating an effective address. When the y bit is set, the upper eight bits are filled with zeroes that cannot be altered.

Data Bank register

The Data Bank register is an 8-bit address register that contains the most significant byte of the effective 24-bit address in all addressing modes. In emulation mode, it contains zeroes, that cannot be altered.

Stack Pointer

In the previous Apple computers, the stack was located at \$100 through \$1FF in memory. In the 65C816, the stack can be located anywhere in bank \$00, but may not exceed 64K. The stack pointer contains the address of the next available stack location. The stack “grows” in a downward direction (toward lower addresses just as with a 6502 stack); push and pull instructions place and remove bytes from the “top” of the stack (actually the lowest address) and grows down.

Program Status register

The Program Status register is an 8-bit register that contains status bits that are set or cleared as a result of the condition of the Accumulator after each operation within the Accumulator. Also, this register contains the e and m bits which control the emulation and native modes. In emulation mode, this register remains unchanged in size.

Program Counter

The Program Counter is a 16-bit register that is concatenated with the Program Bank register to obtain the resulting 24-bit address of the next instruction to be fetched for execution. In emulation mode, this register remains unchanged in size. (See the Program Bank register address description, below).

Program Bank register

The Program Bank register is an 8-bit register that contains the most significant byte of the 24-bit program counter address. In emulation mode, this register is available, although limited in its use, and remains unchanged in size. (See the Program Counter description, above).

Direct register

In the previous Apple computers, the Zero Page (called the Direct Page in the 65C816) was located in the low \$100 bytes of memory, and could not be moved. In the 65C816, the Direct Page can be located anywhere in bank \$00. The starting (low byte) address of the Direct Page is determined by the Direct register. This address can be any value from \$0000 through \$FF00. Although the Direct page can begin anywhere in bank \$00, there is a one-cycle penalty when it does not begin on a page boundary (when the low byte of the Direct register is not \$00).

Emulating the 65C02

As mentioned earlier, the 65C816 is capable of emulating a 65C02 microprocessor. In emulation mode, the 65C816 will execute the complete 65C816 instruction set (which includes all 65C02 instructions), but many of these instructions will be of limited use due to the reduced width of the registers. For instance, addresses are 24 bits wide in native mode but are limited to 16 bits in emulation mode, and data registers that 16 bits wide in native mode are reduced to 8 bits. Note in figure 11-2 that certain bits in some of the registers are filled with specific values that cannot be altered when the m bit is set.

To emulate the 65C02 microprocessor, set the e bit to 1. You may then run programs that were written for the 65C02.

The e bit

The e bit in the Status register controls whether the 65C816 functions like a 65C02 (emulation mode) or like a 65C816 (native mode). When this bit is a 1, the 65C816 executes only those instructions within the 65C02 microprocessor.

In emulation mode, the microprocessor addresses 64K of memory, ignoring the Program Bank register. In native mode, the microprocessor addresses up to 8 megabytes by using the Program Bank register.

The m bit

The m bit in the Status register controls whether the accumulator and memory locations are 8 or 16 bits wide. When the m bit is set, references to the accumulator and memory locations are 8-bits wide. When the m bit is cleared, references to the accumulator and memory locations are 16-bits wide.

In emulation mode (e bit = 1) the m bit is forced to 1, and all references to the accumulator and memory are 8-bits wide.

The x bit

The x bit in the Status Register controls whether the x and y registers are 16 bits or 8 bits wide. When the x bit is set, references to the x and y registers are 8 bits wide. When the x bit is cleared, references to the x and y registers are 16 bits wide.

Operating speed

The Apple Cortland can run the 65C816 processor at one of two speeds: 1MHz and 2.5MHz. The FPI controls the clock input signal to the microprocessor, and selects the appropriate speed as indicated by the Clock Speed bit in the Configuration register.


Summary

The new 65C816 16-bit microprocessor provides these improvements over the 6502:

- 16-bit accumulator
- 16-bit X and Y index registers
- Relocatable zero page
- Relocatable stack
- 24-bit internal address bus
- 8-bit data address bank register
- 8-bit program address bank register
- 11 new addressing modes
- 36 new instructions, for a total of 91 (all 256 op codes)
- Fast block-move instructions
- Ability to emulate 6502 and 65C02 8-bit microprocessors

Cortland Hardware Reference Manual

PLACE 65C816 DATA SHEET HERE



Appendix A



**Roadmap to the Apple IIGS
Technical Manuals**

The Apple IIGs personal computer has many advanced features, making it more complex than earlier models of the Apple II. To describe it fully, Apple has produced a suite of technical manuals. Depending on the way you intend to use the Apple IIGs, you may need to refer to a select few of the manuals, or you may need to refer to most of them.

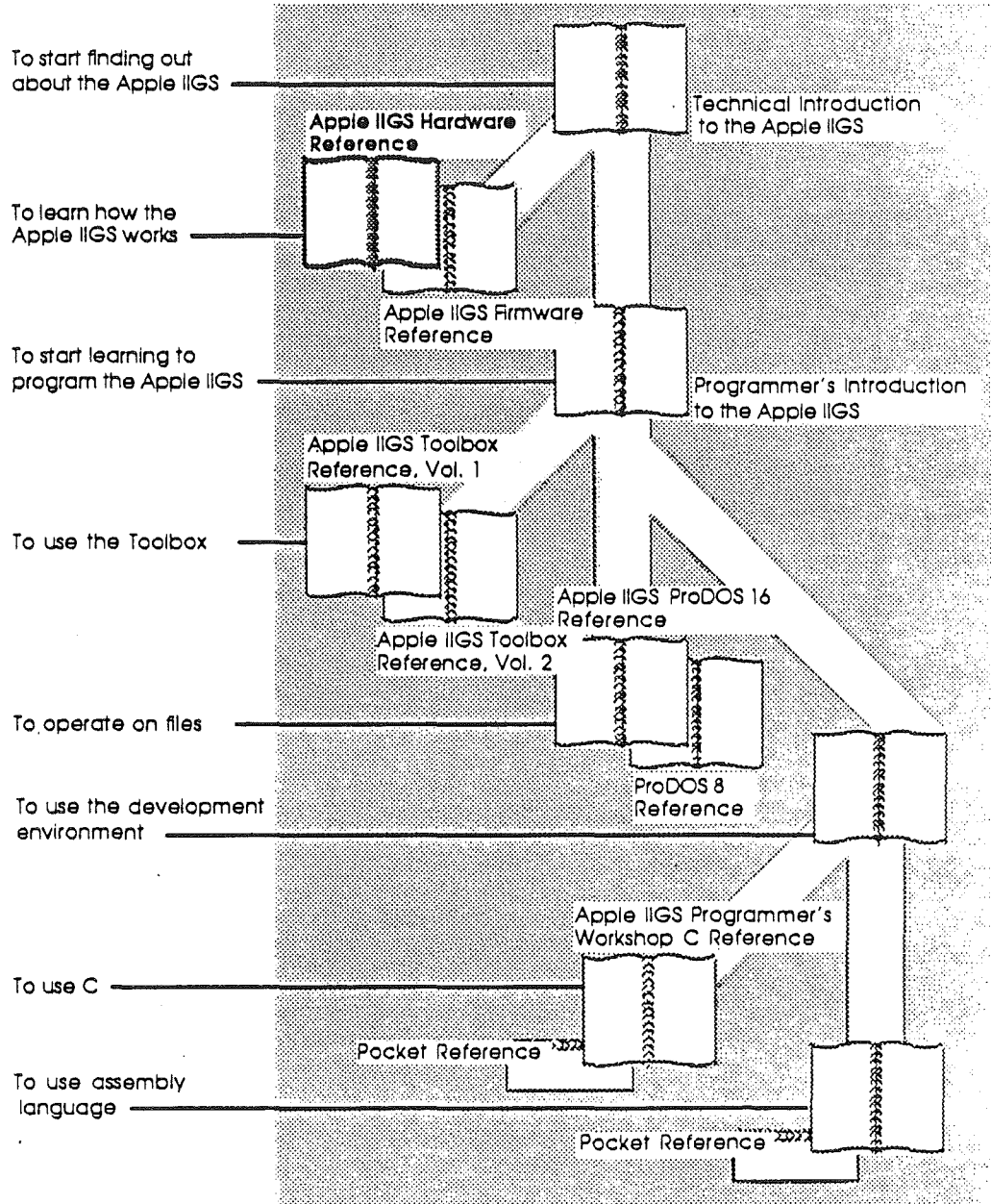
The technical manuals are listed in Table A-1. Figure A-1 is a diagram showing the relationships among the different manuals.

Table A-1
The Apple IIGs technical manuals

Title	Subject
<i>Technical Introduction to the Apple IIGs</i>	What the Apple IIGs is
<i>Apple IIGs Hardware Reference</i>	Machine internals—hardware
<i>Apple IIGs Firmware Reference</i>	Machine internals—firmware
<i>Programmer's Introduction to the Apple IIGs</i>	Concepts and a sample program
<i>Apple IIGs Toolbox Reference: Volume 1</i>	How the tools work and some toolbox specifications
<i>Apple IIGs Toolbox Reference: Volume 2</i>	More toolbox specifications
<i>Apple IIGs Programmer's Workshop Reference</i>	The development environment
<i>Apple IIGs Workshop Assembler Reference*</i>	Using the APW assembler
<i>Apple IIGs Workshop C Reference*</i>	Using C on the Apple IIGs
<i>ProDOS 8 Reference</i>	ProDOS for Apple II programs
<i>Apple IIGs ProDOS 16 Reference</i>	ProDOS and Loader for Apple IIGs
<i>Human Interface Guidelines</i>	Guidelines for the desktop interface
<i>Apple Numerics Manual</i>	Numerics for all Apple computers

*There is a Pocket Reference for each of these.

Figure A-1
Roadmap to the technical manuals



Introductory manuals

These books are introductory manuals for developers, computer enthusiasts, and other Apple IIGS owners who need technical information. As introductory manuals, their purpose is to help the technical reader understand the features of the Apple IIGS, particularly the features that are different from other Apple computers. Having read the introductory manuals, the reader will refer to specific reference manuals for details about a particular aspect of the Apple IIGS.

The technical introduction

The *Technical Introduction to the Apple IIGS* is the first book in the suite of technical manuals about the Apple IIGS. It describes all aspects of the Apple IIGS, including its features and general design, the program environments, the toolbox, and the development environment.

Where the *Apple IIGS Owner's Guide* is an introduction from the point of view of the user, the *Technical Introduction* describes the Apple IIGS from the point of view of the program. In other words, it describes the things the programmer has to consider while designing a program, such as the operating features the program uses and the environment in which the program runs.

The programmer's introduction

When you start writing programs that use the Apple IIGS user interface (with windows, menus, and the mouse), the *Programmer's Introduction to the Apple IIGS* provides the concepts and guidelines you need. It is not a complete course in programming, only a starting point for programmers writing applications for the Apple IIGS. It introduces the routines in the Apple IIGS Toolbox and the program environment they run under. It includes a sample **event-driven program** that demonstrates how a program uses the Toolbox and the operating system.

An **event-driven program** waits in a loop until it detects an event such as a click of the mouse button.

Machine reference manuals

There are two reference manuals for the machine itself: the *Apple IIGS Hardware Reference* and the *Apple IIGS Firmware Reference*. These books contain detailed specifications for people who want to know exactly what's inside the machine.

The hardware reference manual

The *Apple IIGS Hardware Reference* is required reading for hardware developers, and it will also be of interest to anyone else who wants to know how the machine works. Information for developers includes the mechanical and electrical specifications of all connectors, both internal and external. Information of general interest includes descriptions of the internal hardware, which provide a better understanding of the machine's features.

The firmware reference manual

The *Apple IIGS Firmware Reference* describes the programs and subroutines that are stored in the machine's read-only memory (ROM), with two significant exceptions: Applesoft BASIC and the toolbox, which have their own manuals. The *Firmware Reference* includes information about interrupt routines and low-level I/O subroutines for the serial ports, the disk port, and for the DeskTop Bus interface, which controls the keyboard and the mouse. The *Firmware Reference* also describes the Monitor, a low-level programming and debugging aid for assembly-language programs.

The toolbox manuals

Like the Macintosh, the Apple IIGS has a built-in toolbox. The *Apple IIGS Toolbox Reference*, Volume 1, introduces concepts and terminology and tells how to use some of the tools. It also tells how to write and install your own tool set. The *Apple IIGS Toolbox Reference*, Volume 2, contains information about the rest of the tools.

Of course, you don't have to use the toolbox at all. If you only want to write simple programs that don't use the mouse, or windows, or menus, or other parts of the **desktop user interface**, then you can get along without the toolbox. However, if you are developing an application that uses the desktop interface, or if you want to use the Super Hi-Res graphics display, you'll find the toolbox to be indispensable.

In applications that use the **desktop user interface**, commands appear as options in pull-down menus, and material being worked on appears in rectangular areas of the screen called windows. The user selects commands or other material by using the mouse to move a pointer around on the screen.

The Programmer's Workshop manual

The development environment on the Apple IIGS is the Apple IIGS Programmer's Workshop (APW). APW is a set of programs that enable developers to create and debug application programs on the Apple IIGS. The *Apple IIGS Programmer's Workshop Reference* includes information about the parts of the workshop that all developers will use, regardless which programming language they use: the shell, the editor, the linker, the debugger, and the utilities. The manual also tells how to write other programs, such as custom utilities and compilers, to run under the APW Shell.

The APW reference manual describes the way you use the workshop to create an application and includes a sample program to show how this is done.

Programming-language manuals

Apple is currently providing a 65C816 assembler and a C compiler. Other compilers can be used with the workshop, provided that they follow the standards defined in the *Apple IIGS Programmer's Workshop Reference*.

There is a separate reference manual for each programming language on the Apple IIGS. Each manual includes the specifications of the language and of the Apple IIGS libraries for the language, and describes how to write a program in that language. The manuals for the languages Apple provides are the *Apple IIGS Workshop Assembler Reference* and the *Apple IIGS Workshop C Reference*.

Operating-system manuals

There are two operating systems that run on the Apple IIGS: ProDOS 16 and ProDOS 8. Each operating system is described in its own manual: *ProDOS 8 Reference* and *Apple IIGS ProDOS 16 Reference*. ProDOS 16 uses the full power of the Apple IIGS and is not compatible with earlier Apple II's. The ProDOS 16 manual includes information about the System Loader, which works closely with ProDOS 16. If you are writing programs for the Apple IIGS, whether as an application programmer or a system programmer, you are almost certain to need the *ProDOS 16 Reference*.

ProDOS 8, previously just called *ProDOS*, is compatible with the models of Apple II that use 8-bit CPUs. As a developer of Apple IIGS programs, you need to use ProDOS 8 only if you are developing programs to run on 8-bit Apple II's as well as on the Apple IIGS.

All-Apple manuals

In addition to the Apple IIGS manuals mentioned above, there are two manuals that apply to all Apple computers: *Human Interface Guidelines* and *Apple Numerics Manual*. If you develop programs for any Apple computer, you should know about those manuals.

The *Human Interface Guidelines* manual describes Apple's standards for the desktop interface of programs that run on Apple computers. If you are writing an application for the Apple IIGS, you should be familiar with the contents of this manual.

The *Apple Numerics Manual* is the reference for the Standard Apple Numeric Environment (SANE), a full implementation of the IEEE standard floating-point arithmetic. The functions of the Apple IIGS SANE tool set match those of the Macintosh SANE package and of the 6502 assembly language SANE software. If your application requires accurate arithmetic, you'll probably want to use the SANE routines in the Apple IIGS. The *Apple IIGS ToolBox Reference* tells how to use the SANE routines in your programs. The *Apple Numerics Manual* is the comprehensive reference for the SANE numerics routines. A description of the version of the SANE routines for the 65C816 is available through the Apple Programmer's and Developer's Association, administered by the A.P.P.L.E. cooperative in Renton, Washington.

❖ *Note:* The address of the Apple Programmer's and Developer's Association is 290 SW 43rd Street, Renton, WA 98055, and the telephone number is (206) 251-6548.



Appendix B



International Keyboards

Apple makes different versions of the Apple IIgs for different countries. The different versions have different keyboards and display characters that reflect the different typing conventions of the different countries. The ADB keyboard on the Apple IIgs is available in the following versions:

- U.S.A. English
- U.K. English
- Canadian
- French
- German
- Italian
- Spanish
- Swedish
- U.S.A. Dvorak

The keyboards on the localized versions of the Apple IIgs are all mechanically the same; that is, the shapes and arrangement of the keys are the same, only the legends are different. The character decodings for the different versions are all stored in the keyboard decoder ROM. Figures A-1 through A-9 show the legends on the different keyboards.

Figure A-1
U.S.A. English keyboard

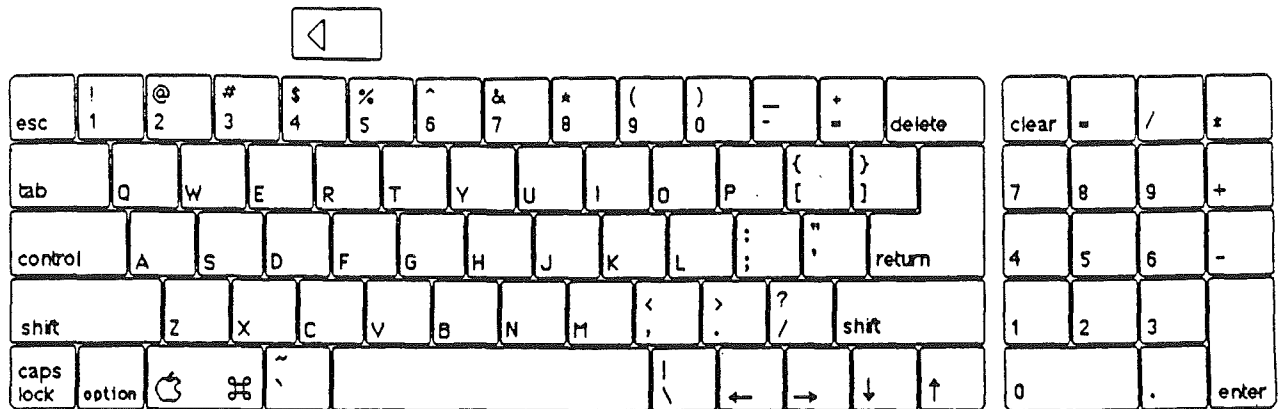


Figure A-2
U.K. English keyboard

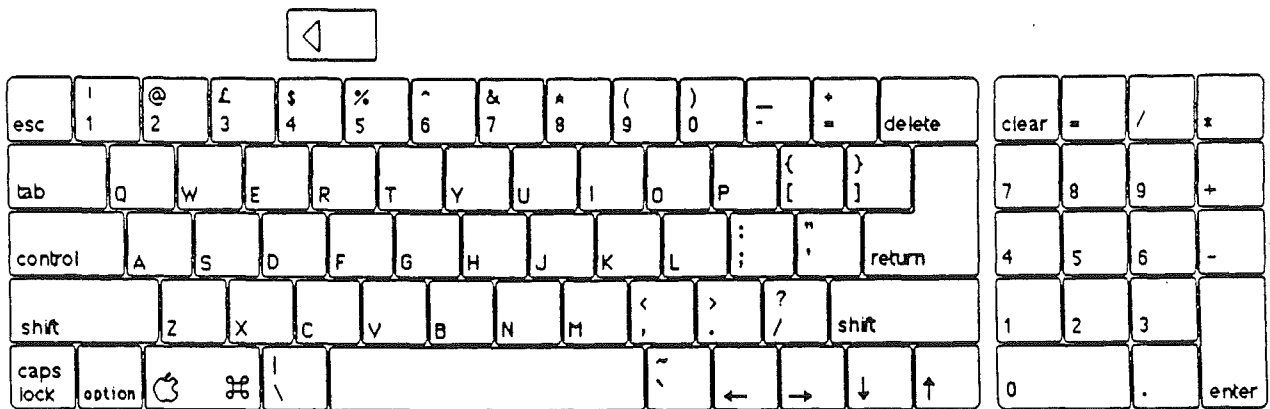


Figure A-3
Canadian keyboard

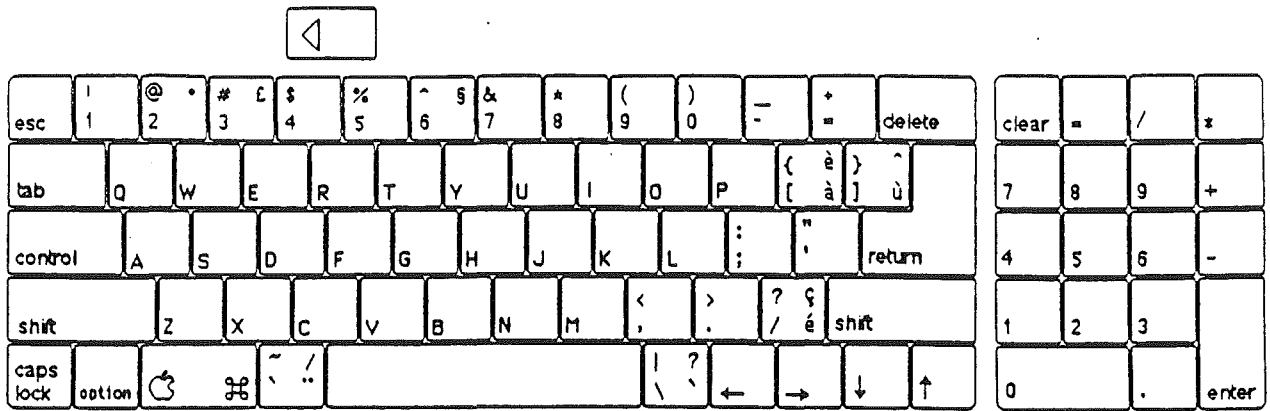


Figure A-4
French keyboard

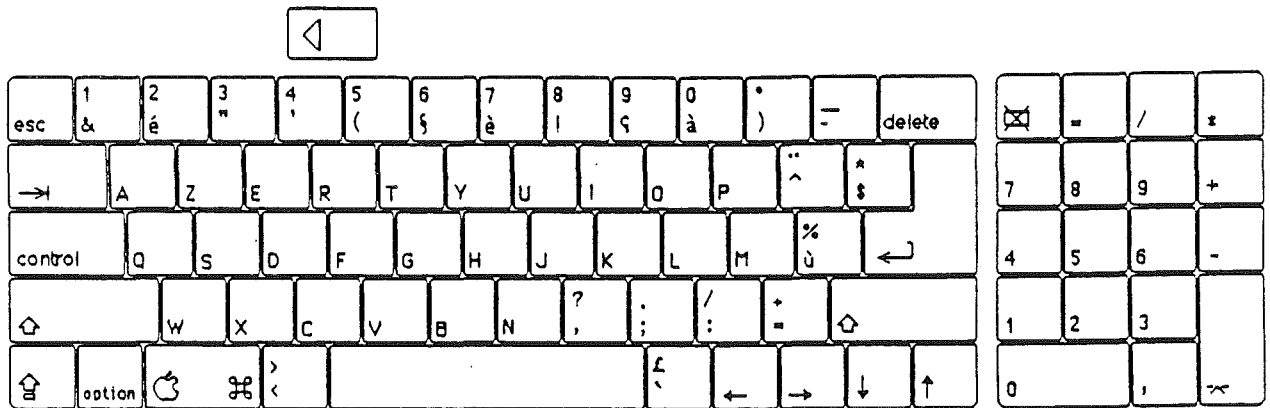


Figure A-5
German keyboard

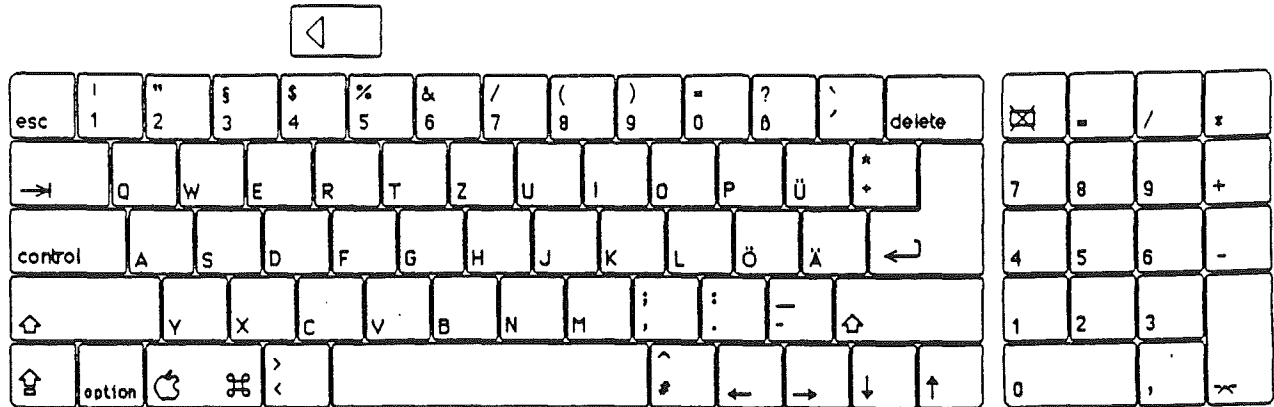


Figure A-6
Italian keyboard

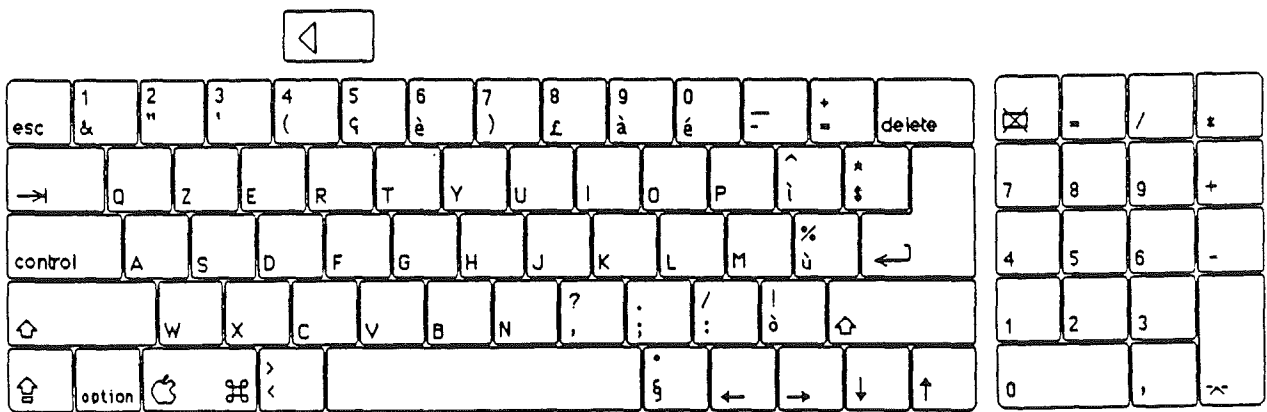


Figure A-7
Spanish keyboard

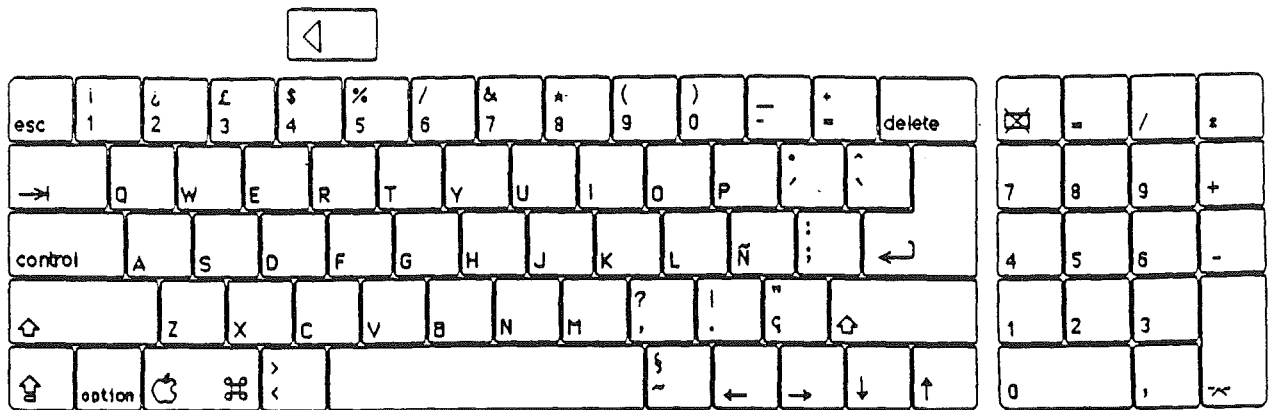


Figure A-8
Swedish keyboard

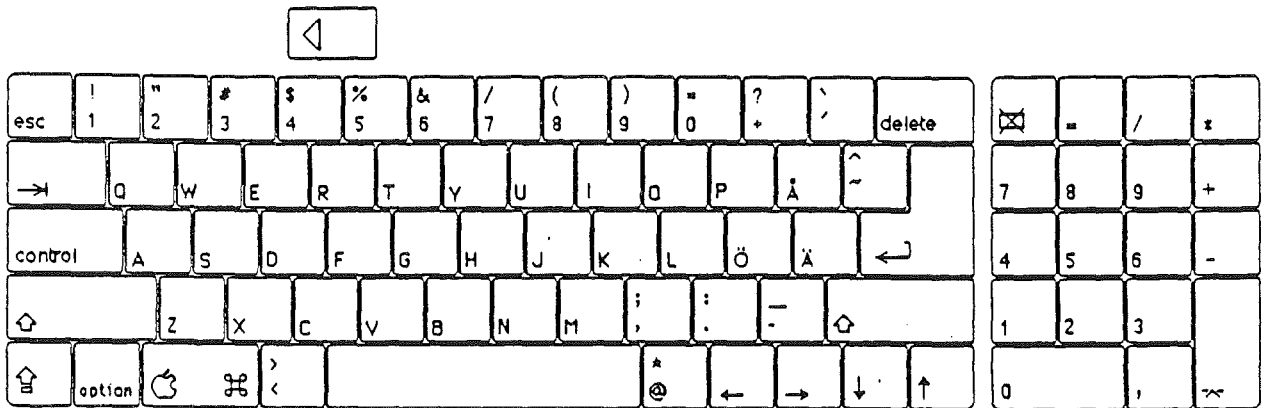
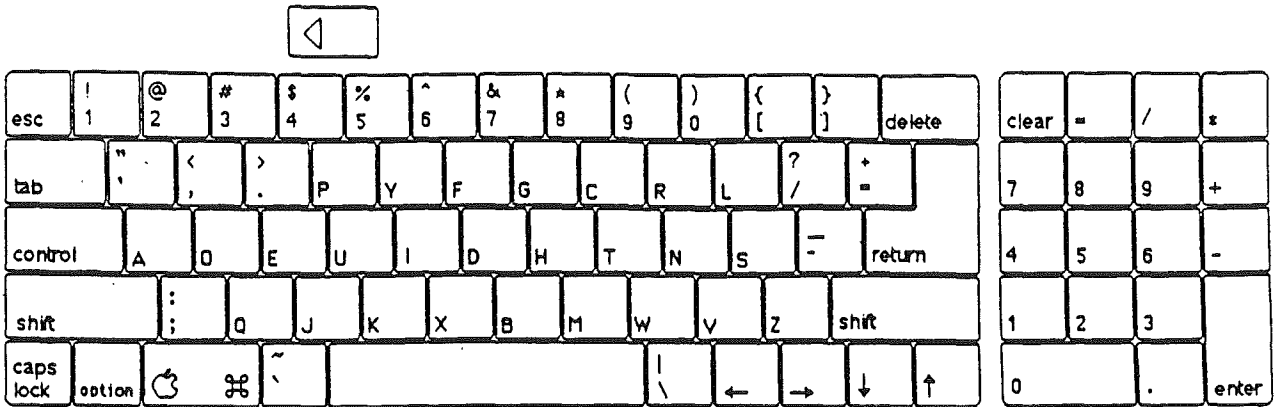
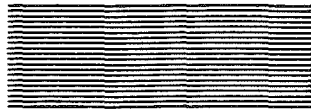


Figure A-9
U.S.A. Dvorak keyboard





Appendix C



Character Generator

This appendix describes the hardware character generator for the 40-column and 80-column text displays. For information about text fonts in Super Hi-Res displays, refer to the QuickDraw II tool set in *Apple IIgs Toolbox Reference*, Volume 1.

Character Generator ROM

The ROM contains the dot patterns making up the characters in the 40-column and 80-column displays.

U.S. Characters

Figures C-1, C-2, C-3 show the characters for the U.S. versions of the computer.

Figure C-1
Uppercase characters

@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\] ^ _

Figure C-2
Lowercase characters

` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

Figure C-3
Special characters

! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?

International characters

For other countries, localized versions of the Apple IIgs substitute appropriate characters for some of the special characters used in the text displays. Figure C-4 shows those characters.

Figure C-4

International characters

Language	Equivalent characters										
U.S.A. English	#	@	[\]	`	{		}	~	
U.K. English	£	@	[\]	`	{		}	~	
French	£	à	"	ç	§	`	é	ù	è	"	
Danish	#	@		Ø	Å	`		ø	å	~	
Spanish	£	§	i	Ñ	¿	`	"	ñ	ç	~	
Italian	£	§	"	ç	é	ù	à	ò	è	ì	
German	#	§	Ä	Ö	Ü	`	ä	ö	ü	ß	
Swedish	#	@	Ä	Ö	Å	`	ä	ö	å	~	

MouseText characters

The character ROM includes several graphic characters used in displaying the desktop user interface in text mode. Figure C-5 shows those characters.

Figure C-5

MouseText Characters

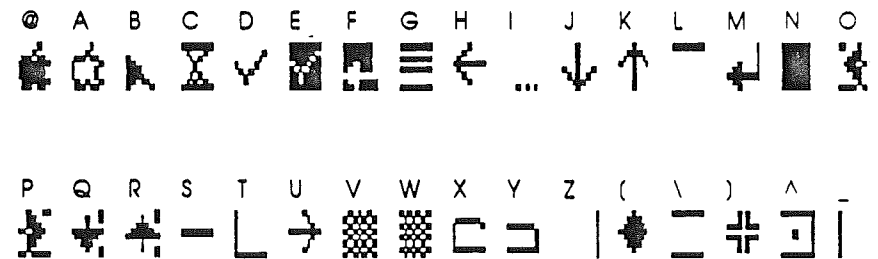
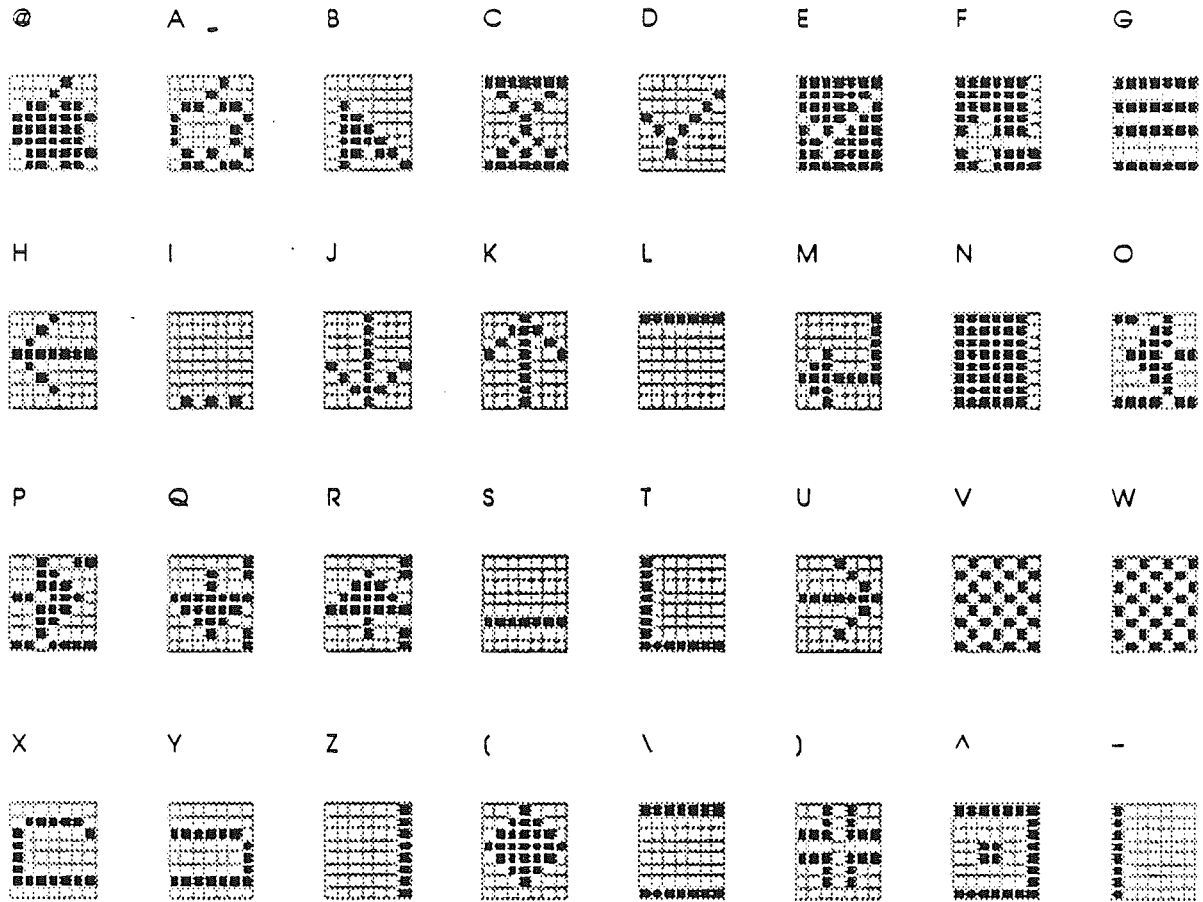


Figure C-5
MouseText characters *** alternate version of this figure ***



Apple IIgs Hardware Reference



Appendix D

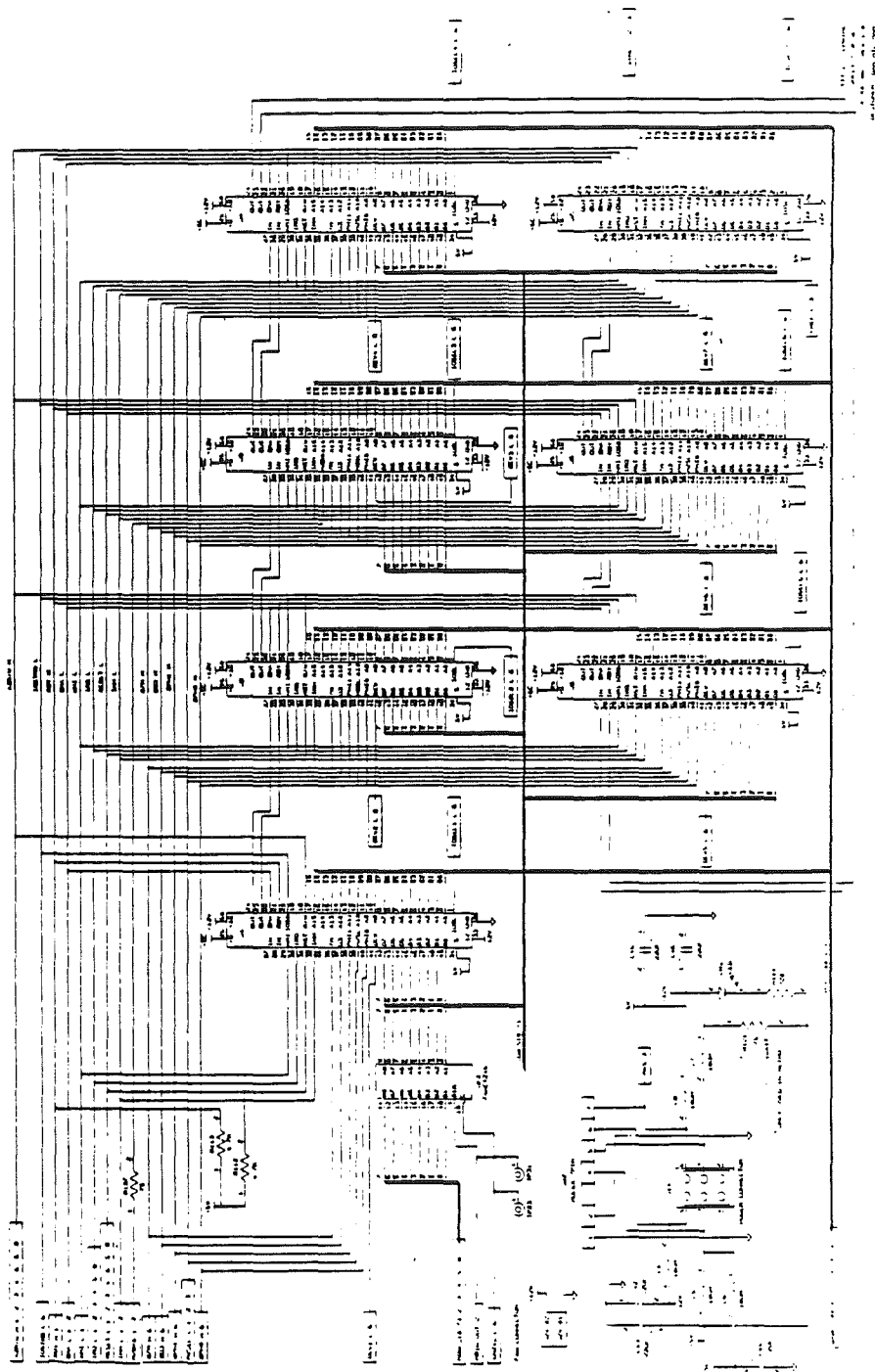


Schematic Diagrams

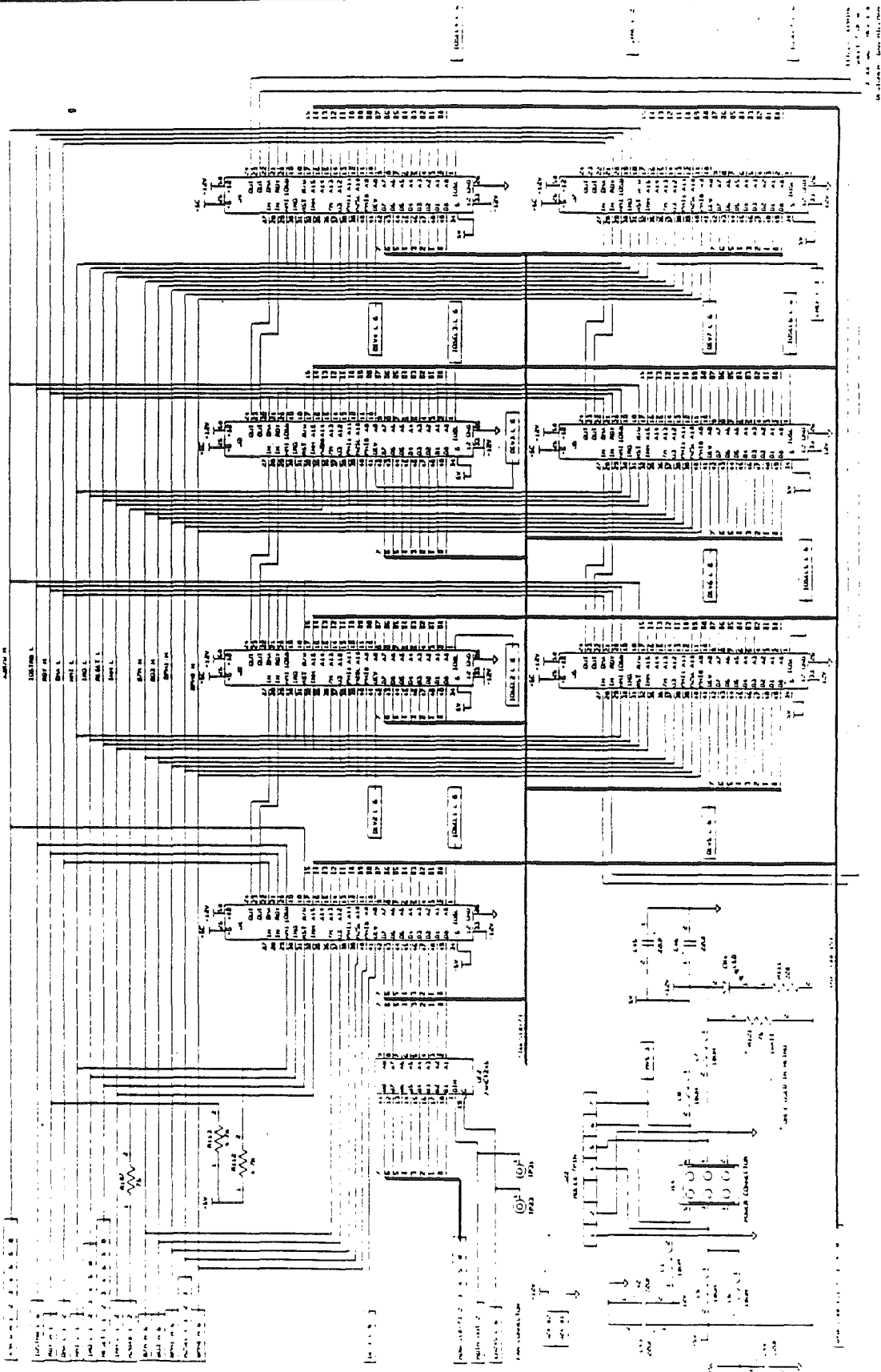
This appendix contains schematic diagrams for the main circuit board of the Apple IIs.

*** For this draft, there is only one of the eight pages of schematics, shown at different degrees of reduction from the original D-size drawing.

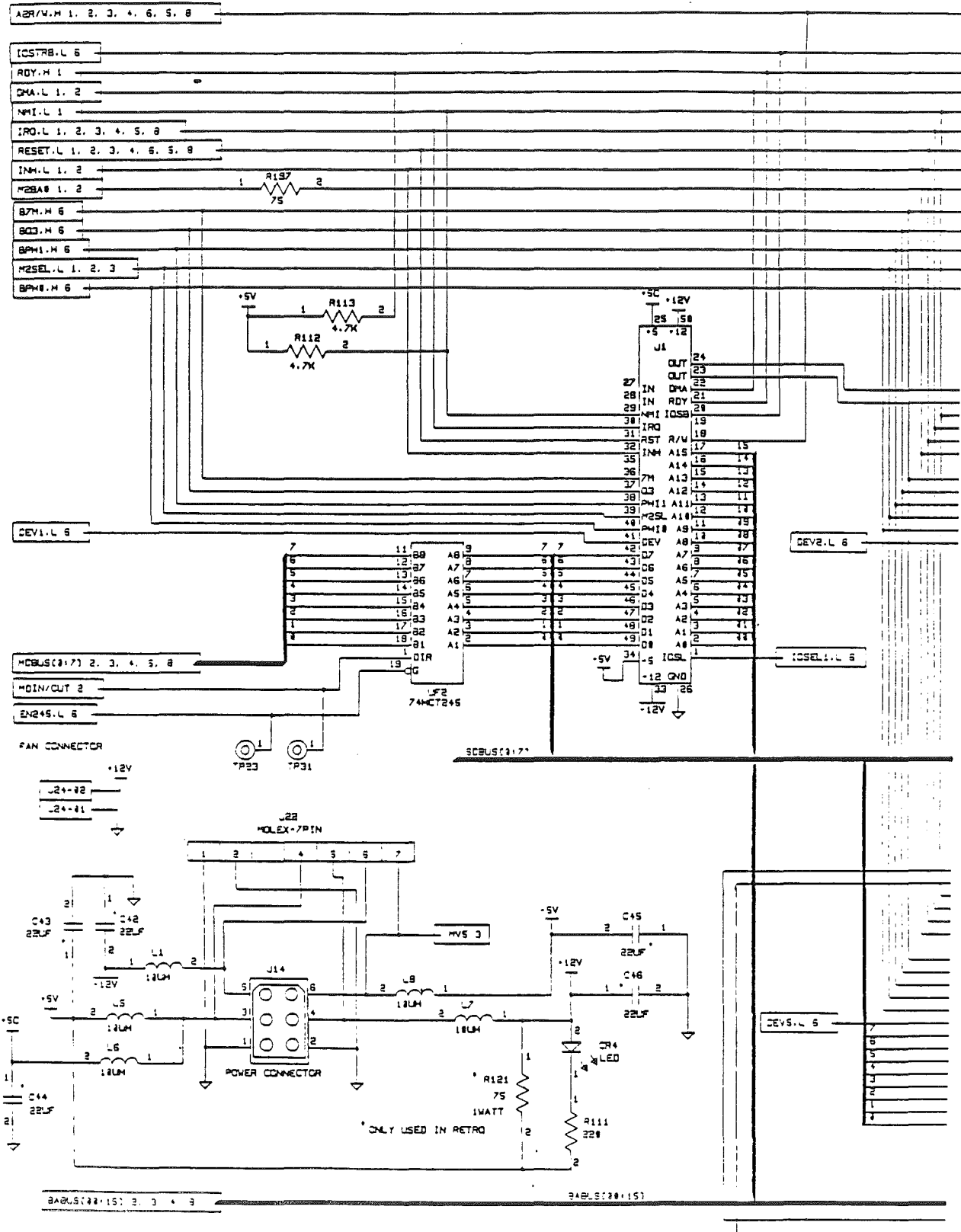
*** Figure D-1. Sample Schematic reduced for 9" page ***



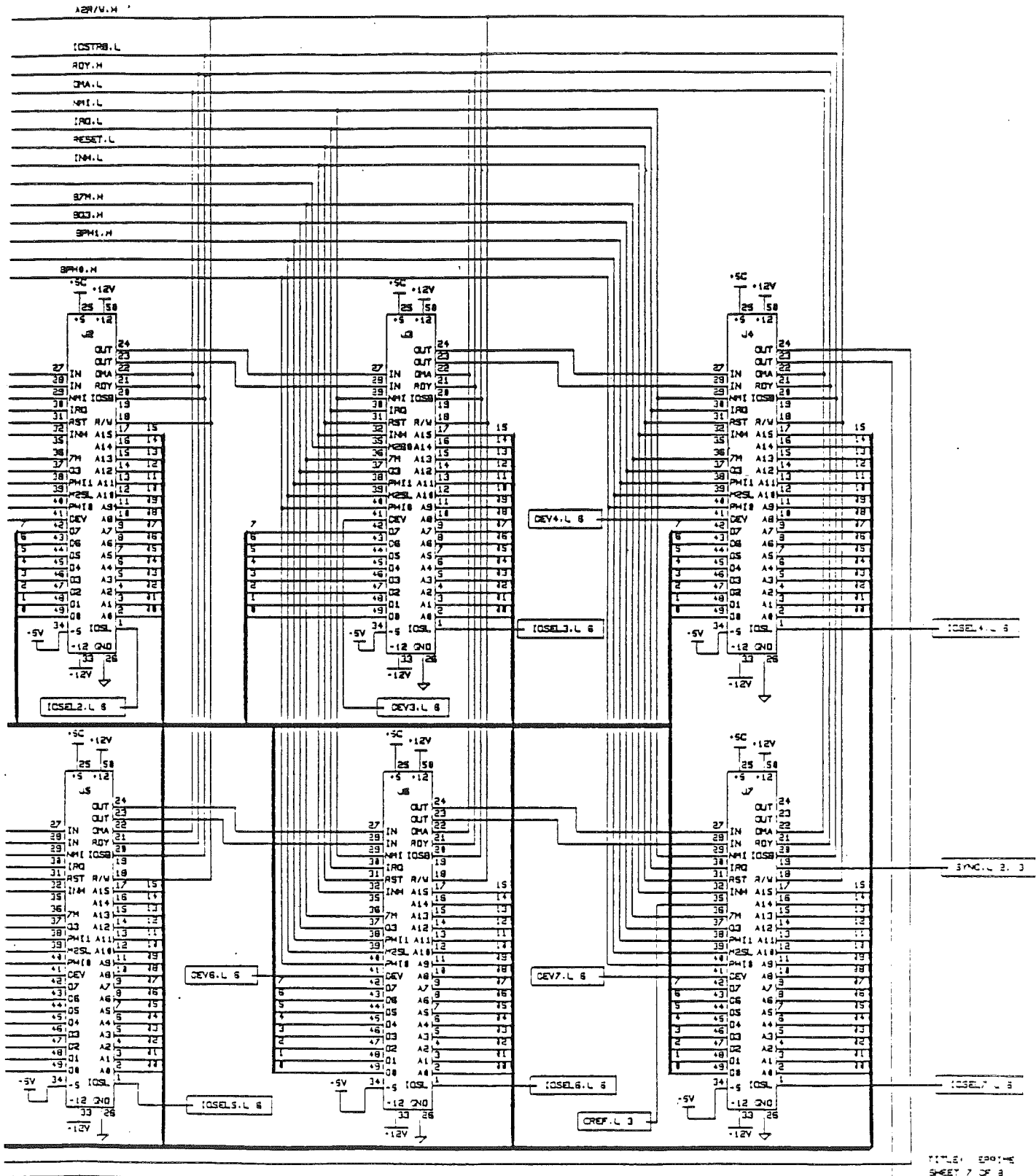
*** Figure D-2. Sample Schematic reduced for 11" page ***



*** Figure D-3. Sample Schematic displayed on two pages: p. 1 ***

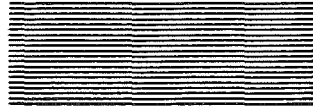


*** Figure D-4. Sample Schematic displayed on two pages: p. 2 ***

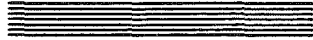


TITLE: EPP014
 SHEET 7 OF 8
 3-JUL-86 REV 1.4

Apple IIgs Hardware Reference



Appendix E



Conversion Tables

This appendix briefly discusses bits and bytes and what they can represent, and peripheral identification numbers. It also contains conversion tables for hexadecimal to decimal and negative decimal, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only.

E.1 Bits and bytes

This section discusses the relationships between bit values and their position within a byte. Here are some rules of thumb regarding the 65C816:

- A bit is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIGS are listed in Table E-1.
- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits comprise a nibble (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and A through F.
- Eight bits (two nibbles) make a byte (figure E-1).
- One byte can represent any of 16 x 16 or 256 values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.
- Each memory location in the Apple IIGS contains one 8-bit byte of data.

- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data. Tables E-5 through E-8 list some of the ways bytes are commonly interpreted.
- Two bytes make a word. The 16 bits of a word can represent any one of 256×256 , or 65536, different values.
- Three bytes make an address. The 24 bits of an address can represent any one of 256×65536 , or 16,777,216, different values.
- The 65C816 uses a 24-bit address to identify a memory location. It can therefore distinguish among 16,777,216 (16M) locations at any given time.
- A memory location is one byte of a 256-byte page. The low-order byte of an address specifies the location in the page. The middle byte specifies the memory page in a 65536-byte (64K) memory bank. The high-order byte specifies which 64K memory bank the byte is in.

Table E-1
What a bit can represent

Context	Representing	0 =	1 =
Binary number	Place value	0	1 x that power of 2
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear*	Set
Serial transfer	Beginning	Start	Carrier (no information yet)
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier
P reg. bit N	Neg. result?	No	Yes
P reg. bit V	Overflow?	No	Yes
P reg. bit B	BRK command?	No	Yes
P reg. bit D	Decimal mode?	No	Yes
P reg. bit I	IRQ interrupts	Enabled	Disabled (masked out)
P reg. bit Z	Zero result?	No	Yes
P reg. bit C	Carry required?	No	Yes

• Sometimes ambiguously termed *reset*.

Figure E-1
Bits, nibbles, and bytes

Binary	Hex	Dec	Binary	Hex	Dec
0000	\$0	0	1000	\$8	8
0001	\$1	1	1001	\$9	9
0010	\$2	2	1010	\$A	10
0011	\$3	3	1011	\$B	11
0100	\$4	4	1100	\$C	12
0101	\$5	5	1101	\$D	13
0110	\$6	6	1110	\$E	14
0111	\$7	7	1111	\$F	15

E.2 Hexadecimal and decimal

Use Table E-2 for conversion of hexadecimal and decimal numbers.

Table E-2
Hexadecimal/Decimal conversion

Digit	\$x000	\$0x00	\$00x0	\$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

Examples:

\$3C = ?	\$FD47 = ?
\$30 = 48	\$F000 = 61440
\$0C = 12	\$ D00 = 3328
\$3C = 60	\$ 40 = 64
	\$ 7 = 7
	\$FD47 = 64839

To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than it. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have 0 left. Add up the hexadecimal numbers.

Example:

16215 = \$?			
16215 - 12288 = 3927	12288 = \$7000	3927 -	
3840 = 87	3840 = \$ F00	87 -	
80 = 7	80 = \$ 50		
7	7 = \$ 7		
	16215 = \$7F57		

E.3 Hexadecimal and negative decimal

If a number is larger than decimal 32767, Applesoft BASIC allows you to use the negative-decimal equivalent of the number. Table E-3 is set up to make it easy for you to convert a hexadecimal number directly to a negative-decimal number.

Table E-3
Hexadecimal to negative decimal conversion

Digit	\$x000	\$\$0x00	\$\$\$00x0	\$\$\$\$000x
	0	0	0	-1
E	-4096	-256	-16	-2
D	-8192	-512	-32	-3
C	-12288	-768	-48	-4
B	-16384	-1024	-64	-5
A	-20480	-1280	-80	-6
9	-24576	-1536	-96	-7
8	-28672	-1792	-112	-8

7	-2048	-128	-9
6	-2304	-144	-10
5	-2560	-160	-11
4	-2816	-176	-12
3	-3072	-192	-13
2	-3328	-208	-14
1	-3584	-224	-15
0	-3840	-240	-16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (0s included). Then add their values (ignoring their signs for a moment). The resulting number, with a minus sign in front of it, is the desired negative-decimal number.

Example:

```

SC010 = - ?
SC000: -12288 $ 000: - 3840 $ 10: - 224 $ 0: -
      16 _____
SC010 -16368
    
```

To convert a negative-decimal number directly to a positive-decimal number, add it to 65536. (This addition ends up looking like subtraction.)

Example:

```

-151 = + ?
65536 + (-151) = 65536 - 151 = 65385
    
```

To convert a negative-decimal number to a hexadecimal number, first convert it to a positive-decimal number, then use Table E-2.

E.4 Peripheral identification numbers

Many Apple products now use peripheral identification numbers (called PIN numbers) as shorthand to designate serial device characteristics. The Apple II series *Universal Utilities* disk presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

Table E-4 is a definition of the PIN number digits. When communication mode is selected, the seventh digit is ignored.

Example: 252/1111 means:

Communication mode	No parity
8 data bits, 1 stop bit	Do not echo output to display
300 baud (bits per second)	No line feed after carriage return Do not generate carriage returns

Table E-4
PIN numbers

x	x	x	/	x	x	x	x
---	---	---	---	---	---	---	---

1 = Printer mode
2 = Communication mode*

1 = 6 data bits, 1 stop bit
2 = 6 data bits, 2 stop bits
3 = 7 data bits, 1 stop bit
4 = 7 data bits, 2 stop bits
5 = 8 data bits, 1 stop bit
6 = 8 data bits, 2 stop bits

1 = 110 bits per second
2 = 300 bits per second
3 = 1200 bits per second
4 = 2400 bits per second
5 = 4800 bits per second
6 = 9600 bits per second
7 = 19200 bits per second

1 = No parity
2 = Even parity (total on = even)
3 = Odd parity (total on = odd)
4 = MARK parity (parity bit = 1)
5 = SPACE parity (parity bit = 0)

1 = Do not echo output on screen
2 = Echo output on screen

1 = Do not generate LF after CR
2 = Generate LF after CR

1 = Do not generate CR*

- 2 = Generate CR after 40 characters
- 3 = Generate CR after 72 characters
- 4 = Generate CR after 80 characters
- 5 = Generate CR after 132 characters

• If you select communication mode, then seventh digit must equal 1. This value is supplied automatically when you use the UUD.

E.5 Eight-Bit code conversions

Tables E-5 through E-8 show the entire ASCII character set. Note that character values are shown with the high bit off. Unless otherwise noted, all ASCII character values above \$7F (127 decimal) generate the same character as that value with the high bit off. Here is how to interpret these tables:

- The *Binary* column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0 (SPACE parity or Pascal)—for example, 01001000 for the letter *E*.
- The last 128 ASCII entries (from 128 through 255) represent 7-bit ASCII codes plus a high-order bit of 1 (MARK parity or BASIC)—for example, 11001000 for the letter *E*.
- A transmitted or received ASCII character will take whichever form (in the communication register) is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity *H*, 01001000 for an even-parity *E*.
- The *ASCII Char* column gives the ASCII character name.
- The *Interpretation* column spells out the meaning of special symbols and abbreviations, where necessary.
- The *What to Type* column indicates what keystrokes generate the ASCII character (where it is not obvious).
- The columns marked *Prt* and *Alt* indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

Note that the values \$40 through \$5F (and \$C0 through \$DF) in the alternate character set are displayed as MouseText characters (Figure 5-1) if the firmware is set to do so (Section 5.2.2), or if the firmware is bypassed.

❖ *Note:* The primary and alternate displayed character sets in Tables E-5 through E-8 are the result of firmware mapping. The

character generator ROM actually contains only one character set. The firmware mapping procedure is described in Section 3.3.6.

Table E-5
Control characters, high bit off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	PH	Alt
0000000	0	\$00	NUL	Blank (null)	Control-@	@	@
0000001	1	\$01	SOH	Start of Header	Control-A	A	A
0000010	2	\$02	STX	Start of Text	Control-B	B	B
0000011	3	\$03	ETX	End of Text	Control-C	C	C
0000100	4	\$04	EOT	End of Transm.	Control-D	D	D
0000101	5	\$05	ENQ	Enquiry	Control-E	E	E
0000110	6	\$06	ACK	Acknowledge	Control-F	F	F
0000111	7	\$07	BEL	Bell	Control-G	G	G
0001000	8	\$08	BS	Backspace	Control-H or Left-Arrow-H	H	H
0001001	9	\$09	HT	Horizontal Tab	Control-I or Tab	I	I
0001010	10	\$0A	LF	Line Feed	Control-J or Down-Arrow-J	J	J
0001011	11	\$0B	VT	Vertical Tab	Control-K or Up-Arrow	K	K
0001100	12	\$0C	FF	Form Feed	Control-L	L	L
0001101	13	\$0D	CR	Carriage Return	Control-M or Return	M	M
0001110	14	\$0E	SO	Shift Out	Control-N	N	N
0001111	15	\$0F	SI	Shift In	Control-O	O	O
0010000	16	\$10	DLE	Data Link Escape	Control-P	P	P
0010001	17	\$11	DC1	Device Control 1	Control-Q	Q	Q
0010010	18	\$12	DC2	Device Control 2	Control-R	R	R
0010011	19	\$13	DC3	Device Control 3	Control-S	S	S
0010100	20	\$14	DC4	Device Control 4	Control-T	T	T
0010101	21	\$15	NAK	Neg. Acknowledge	Control-U or Right-Arrow	U	U
0010110	22	\$16	SYN	Synchronization	Control-V	V	V
0010111	23	\$17	ETB	End of Text Blk.	Control-W	W	W
0011000	24	\$18	CAN	Cancel	Control-X	X	X
0011001	25	\$19	EM	End of Medium	Control-Y	Y	Y
0011010	26	\$1A	SUB	Substitute	Control-Z	Z	Z
0011011	27	\$1B	ESC	Escape	Control-[or Escape	[[
0011100	28	\$1C	FS	File Separator	Control-\	\	\
0011101	29	\$1D	GS	Group Separator	Control-]]]
0011110	30	\$1E	RS	Record Separator	Control-^	^	^
0011111	31	\$1F	US	Unit Separator	Control-_	_	_

Table E-6
Special characters, high bit off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
0100000	32	\$20	SP	Space	Space bar		
0100001	33	\$21	!			!	!
0100010	34	\$22	"			"	"
0100011	35	\$23	#			#	#
0100100	36	\$24	\$			\$	\$
0100101	37	\$25	%			%	%
0100110	38	\$26	&			&	&
0100111	39	\$27	'	Apostrophe		'	'
0101000	40	\$28	(((
0101001	41	\$29)))
0101010	42	\$2A	*			*	*
0101011	43	\$2B	+			+	+
0101100	44	\$2C	,	Comma		,	,
0101101	45	\$2D	-	Hyphen		-	-
0101110	46	\$2E	.	Period		.	.
0101111	47	\$2F	/			/	/
0110000	48	\$30	0			0	0
0110001	49	\$31	1			1	1
0110010	50	\$32	2			2	2
0110011	51	\$33	3			3	3
0110100	52	\$34	4			4	4
0110101	53	\$35	5			5	5
0110110	54	\$36	6			6	6
0110111	55	\$37	7			7	7
0111000	56	\$38	8			8	8
0111001	57	\$39	9			9	9
0111010	58	\$3A	:			:	:
0111011	59	\$3B	;			;	;
0111100	60	\$3C	<			<	<
0111101	61	\$3D	=			=	=
0111110	62	\$3E	>			>	>
0111111	63	\$3F	?			?	?

Table E-7
Uppercase characters, high bit off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt*
1000000	64	\$40	@			@	
1000001	65	\$41	A			A	
1000010	66	\$42	B			B	
1000011	67	\$43	C			C	

Appendix E: Conversion Tables

1000100	68	\$44	D		D
1000101	69	\$45	E		E
1000110	70	\$46	F		F
1000111	71	\$47	G		G
1001000	72	\$48	H		H
1001001	73	\$49	I		I
1001010	74	\$4A	J		J
1001011	75	\$4B	K		K
1001100	76	\$4C	L		L
1001101	77	\$4D	M		M
1001110	78	\$4E	N		N
1001111	79	\$4F	O		O
1010000	80	\$50	P		P
1010001	81	\$51	Q		Q
1010010	82	\$52	R		R
1010011	83	\$53	S		S
1010100	84	\$54	T		T
1010101	85	\$55	U		U
1010110	86	\$56	V		V
1010111	87	\$57	W		W
1011000	88	\$58	X		X
1011001	89	\$59	Y		Y
1011010	90	\$5A	Z		Z
1011011	91	\$5B	[Opening bracket	/
1011100	92	\$5C	\	Reverse slant	\
1011101	93	\$5D]	Closing bracket	/
1011110	94	\$5E	^	Caret	^
1011111	95	\$5F	_	Underline	_

* If the high bit is set, the MouseText characters are replaced with their equivalent in the primary character set with that value.

Table E-8
Lowercase characters, high bit off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Prt	Alt
1100000	96	\$60	`	Grave accent			`
1100001	97	\$61	a			!	a
1100010	98	\$62	b			"	b
1100011	99	\$63	c			#	c
1100100	100	\$64	d			\$	d
1100101	101	\$65	e			%	e
1100110	102	\$66	f			&	f
1100111	103	\$67	g			'	g
1101000	104	\$68	h			(h

Apple IIGS Hardware Reference

1101001	105	\$69	i)	i
1101010	106	\$6A	j	*	j
1101011	107	\$6B	k	+	k
1101100	108	\$6C	l	,	l
1101101	109	\$6D	m	-	m
1101110	110	\$6E	n	.	n
1101111	111	\$6F	o	/	o
1110000	112	\$70	p	0	p
1110001	113	\$71	q	1	q
1110010	114	\$72	r	2	r
1110011	115	\$73	s	3	s
1110100	116	\$74	t	4	t
1110101	117	\$75	u	5	u
1110110	118	\$76	v	6	v
1110111	119	\$77	w	7	w
1111000	120	\$78	x	8	x
1111001	121	\$79	y	9	y
1111010	122	\$7A	z	:	z
1111011	123	\$7B	{	;	{
1111100	124	\$7C		<	
1111101	125	\$7D	}	=	}
1111110	126	\$7E	~	>	~
1111111	127	\$7F	DEL	?	DE

L



Appendix F



Frequently Used Tables

This appendix contains frequently-used tables from throughout the manual.

Apple IIgs Hardware Reference

Glossary

This glossary defines technical terms used in this book. Boldfaced terms within a definition are also defined in the glossary.

accumulator: The register in a computer's central processor or microprocessor where most computations are performed.

ACIA: Acronym for *Asynchronous Communications Interface Adapter*, a type of communications IC used in some Apple computers. See **SCC**.

acronym: A word formed from the initial letters of a name or phrase, such as ROM (from *read-only memory*).

ADC: See **analog-to-digital converter**.

address: A number that specifies the location of a single byte of memory. Addresses can be given as decimal integers or as hexadecimal integers. A 64K system has addresses ranging from 0 to 65535 (in decimal) or from \$0000 to \$FFFF (in hexadecimal). The letter X in an address stands for all possible values for that digit. For example, \$Dxxx means all the addresses from \$D000 through \$DFFF.

American Simplified Keyboard: See **Dvorak keyboard**.

American Standard Code for Information Interchange: See **ASCII**.

analog: (adj) Varying smoothly and continuously over a range, rather than changing in discrete jumps. For example, a conventional 12-hour clock face is an analog device that shows the time of day by the continuously changing position of the clock's hands. Compare **digital**.

analog RGB: A type of color video monitor that accepts separate analog signals for the red, green, and blue color primaries. The intensity of each primary can vary continuously, making possible many shades and tints of color.

analog signal: A signal that varies continuously over time, rather than being sent and received in discrete intervals. Compare **digital signal**.

analog-to-digital converter (ADC): A device that converts quantities from analog to digital form. For example, computer hand controls convert the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes stepwise even when the dial is turned smoothly.

Apple key: A modifier key on the Apple II GS keyboard, marked with both an Apple icon and a spinner, the icon used on the equivalent key on some Macintosh keyboards. See **Open Apple**.

AppleTalk: Apple's local-area network for Apple II and Macintosh and the LaserWriter and ImageWriter II. Like the Macintosh, the Apple II GS has the AppleTalk interface built in.

Apple IIGS Hardware Reference

AppleTalk connector: A piece of equipment, consisting of a connection box, a short cable, and an 8-pin miniature DIN connector, that enables a Apple IIGS to be part of an AppleTalk network.

Apple II: A family of computers, including the original Apple II, the Apple II Plus, the Apple IIe, the Apple IIc, and the Apple IIGS.

Apple IIc: A transportable personal computer in the Apple II family, with a disk drive and 80-column display capability built in.

Apple IIe: A personal computer in the Apple II family with seven expansion slots and an auxiliary memory slot that allow the user to enhance the computer's capabilities with peripheral and auxiliary cards.

Apple IIe 80-Column Text Card: A peripheral card that plugs into the Apple IIe's auxiliary memory slot and enables the computer to display text as either 40 or 80 characters per line.

Apple IIe Extended 80-Column Text Card: A peripheral card that plugs into the Apple IIe's auxiliary memory slot and enables the computer to display text as either 40 or 80 characters per line while extending the computer's memory capacity by 64K.

Apple II Plus: A personal computer in the Apple II family with expansion slots that allow the user to enhance the computer's capabilities with peripheral and auxiliary cards.

ASCII: Acronym for *American Standard Code for Information Interchange*, pronounced *ASK-ee*. A code in which the numbers from 0 to 127 stand for text characters. ASCII code is used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

aspect ratio: The ratio of an image's width to its height. For example, a standard video display has an aspect ratio of 4:3.

asynchronous: Not synchronized by a mutual timing signal or clock. Compare **synchronous**.

Asynchronous Communications Interface Adapter: See ACIA.

auxiliary slot: The special expansion slot inside the Apple IIe used for the Apple IIe 80-Column Text Card or Extended 80-Column Text Card, and also for the RGB monitor card. The slot is labeled AUX. CONNECTOR on the circuit board.

back panel: The rear surface of the computer, which includes the power switch, the power connector, and connectors for peripheral devices.

baud: A unit of data transmission speed: the number of discrete signal state changes per second. Often, but not always, equivalent to *bits per second*. Compare **bit rate**.

bit: A contraction of *binary digit*. The smallest unit of information that a computer can hold. The value of a bit (1 or 0) represents a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

bit image: A collection of bits in memory that have a rectilinear graphical representation. The display on the screen is a visible bit image.

bitmap: A set of bits that represents the positions and states of a corresponding set of items; for example, dots in an image. See **bit image**.

bit rate: The speed at which bits are transmitted, usually expressed as *bits per second*, or *bps*. Compare **baud**.

block I/O device: A type of device that reads or writes information in organized groups called blocks, which are typically 512 bytes long. A disk drive is a block device.

boot: Another way to say **start up**. A computer boots by loading a program into memory from an external storage medium such as a disk. *Boot* is short for *bootstrap load*, a term suggestive of the difficulty of initial loading of loader programs into early computers that didn't have built-in firmware in ROM.

bootstrap: See **boot**.

buffer: A holding area in the computer's memory where information can be stored by one program or device and then read at a different rate by another; for example, a print buffer.

bus: A group of wires or circuits that transmit related information from one part of a computer system to another. In a network, a line of cable with connectors linking devices together. A bus network has a beginning and an end. (It's not in a closed circle or T shape.)

byte: A unit of measure of computer data or **memory**, consisting of a fixed number of **bits**. On Apple II systems, one byte consists of eight bits, and a byte can have any value between 0 and 255. The value can represent an instruction, letter, number, punctuation mark, or other character. See also **kilobyte**, **megabyte**.

carriage return: An ASCII character (decimal 13) that ordinarily causes a printer or display device to place the next character on the left margin.

carry flag: A status bit in the microprocessor, used as an additional high-order bit with the accumulator bits in addition, subtraction, rotation, and shift operations.

cathode-ray tube: A display device.

central processing unit (CPU): The part of the computer that performs the actual computations in machine language. See **microprocessor**.

character: Any symbol that has a widely understood meaning and thus can convey information. Some characters—such as letters, numbers, and punctuation—can be displayed on the monitor screen and printed on a printer.

chip: See **integrated circuit**.

circuit board: A board containing embedded circuits and an attached collection of integrated circuits (chips).

clock chip: A special chip in which parameter RAM and the current setting for the date and time are stored. This chip is powered by a battery when the system is off, thus preserving the information.

CMOS: Abbreviation for *complementary metal oxide silicon*, one of several methods of making integrated circuits out of silicon. CMOS devices are characterized by their low power consumption. CMOS techniques are derived from MOS techniques.

code: (1) A number or symbol used to represent some piece of information. (2) The statements or instructions that make up a program.

column: A vertical arrangement of graphics points or character positions on the display.

component: A part; in particular, a part of a computer system.

composite video: A video signal that includes both display information and the synchronization (and other) signals needed to display it. See **NTSC**, **RGB monitor**.

computer: An electronic device that performs predefined (programmed) computations at high speed and with great accuracy. A machine that is used to store, transfer, and transform information.

computer language: See **programming language**.

configuration: (1) The total combination and arrangement of hardware components—CPU, video display device, keyboard, and peripheral devices—that make up a computer system. (2) The software settings that allow various hardware components of a computer system to communicate with each other.

Control key: A specific modifier key on Apple II-family keyboards that produces control characters when used in combination with other keys.

control registers: Special registers that programs can read and write, similar to soft switches. The control registers are specific locations in the I/O space (\$Cxxx) in bank \$E0; they are accessible from bank \$00 if I/O shadowing is on.

controller card: A peripheral card that connects a device such as a printer or disk drive to a computer's main logic board and controls the operation of the device.

CPU: See **central processing unit**.

cursor: A symbol displayed on the screen marking where the user's next action will take effect or where the next character typed from the keyboard will appear.

DAC: See **digital-to-analog converter**.

data: information transferred to or from or stored in a computer or other mechanical communications or storage device.

data bits: The bits in a communication transfer that contain information. Compare **start bit**, **stop bit**.

data format: The form in which data is stored, manipulated, or transferred. For example, when data is transmitted and received serially, it typically has a data format of one start bit, five to eight data bits, an optional parity bit, and one or two stop bits.

Data Carrier Detect (DCD): A signal from a DCE (such as a modem) to a DTE (such as an Apple IIc) indicating that a communication connection has been established. See **Data Communication Equipment, Data Terminal Equipment.**

Data Communication Equipment (DCE): As defined by the RS-232-C standard, any device that transmits or receives information. Usually this device is a modem.

Data Set Ready (DSR): A signal from a DCE to a DTE indicating that the DCE has established a connection. See **Data Communication Equipment, Data Terminal Equipment.**

Data Terminal Equipment (DTE): As defined by the RS-232-C standard, any device that generates or absorbs information, thus acting as an endpoint of a communication connection. A computer might serve as a DTE.

Data Terminal Ready (DTR): A signal from a DTE to a DCE indicating a readiness to transmit or receive data. See **Data Communication Equipment, Data Terminal Equipment.**

DCD: See **Data Carrier Detect.**

DCE: See **Data Communication Equipment.**

Delete key: A key on the upper-right corner of the Apple IIe, Apple IIc, and Apple IIGS keyboards that erases the character immediately preceding (to the left of) the cursor. Similar to the Macintosh Backspace key.

digital: (adj) Represented in a discrete (noncontinuous) form, such as numerical digits or integers. For example, contemporary digital clocks show the time as a digital display (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog.**

digital oscillator chip: an integrated circuit that contains thirty-two digital oscillators, each of which can generate a sound from stored digital waveform data.

digital signal: A signal that is sent and received in discrete intervals. A signal that does not vary continuously over time. Compare **analog signal.**

digital-to-analog converter: A device that converts quantities from digital to analog form.

DIN: Abbreviation for *Deutsche Industrie Normal*, a European standards organization.

DIN connector: A type of connector with multiple pins inside a round outer shield.

direct page: A page (256 bytes) of memory in the Apple IIGS that works like the zero page in a 6502 system but can reside anywhere in bank \$00, rather than always starting at location \$0000. Co-resident programs or routines can have their own direct pages at different locations.

disk controller card: A peripheral card that provides the connection between one or two disk drives and the computer. (This connection, or interface, is built into the Apple IIc, the Apple IIGS, and all Macintosh-family computers.)

Disk II drive: An older type of disk drive made and sold by Apple Computer for use with the Apple II, II Plus, and IIe. It uses 5.25-inch floppy disks.

display: (1) A general term to describe what you see on the screen of your display device when you're using a computer. (2) Short for a display device.

display device: A device that displays information, such as a television set or video monitor.

dithering: A technique for alternating the values of adjacent pixels to create the effect of intermediate values. Dithering can give the effect of shades of gray on a black-and-white display, or more colors on a color display.

DOC: See **digital oscillator chip**.

DSR: See **Data Set Ready**.

DTE: See **Data Terminal Equipment**.

DTR: See **Data Terminal Ready**.

Dvorak keyboard: An alternate keyboard layout, also known as the *American Simplified Keyboard*, which increases typing speed because the keys most often used are in the positions easiest to reach. Compare **QWERTY keyboard**.

e flag: One of three flag bits in the 65C816 processor that programs use to control the processor's operating modes. The setting of the e flag determines whether the processor is in native mode or emulation mode. See **m flag**, **x flag**.

effective address: In machine-language programming, the address of the memory location on which a particular instruction operates, which may be arrived at by indexed addressing or some other addressing method.

8-bit Apple II: Another way of saying standard Apple II, that is, any Apple II with an 8-bit microprocessor (6502 or 65C02).

80-column text card: A peripheral card that allows the Apple II, Apple II Plus, and Apple IIe to display text in 80 columns (in addition to the standard 40 columns).

emulate: To operate in a way identical to a different system. For example, the 65C816 microprocessor in the Apple IIGS can carry out all the instructions in a program originally written for an Apple II that uses a 6502 microprocessor, thus emulating the 6502.

emulation mode: A manner of operating in which one system imitates another. In the Apple IIGS, the mode the 65C816 is in when the Apple IIGS is running programs written for Apple II's that use the 6502.

Escape character: An ASCII character that, with many programs and devices, allows you to perform special functions when used in combination keypresses.

Escape key: A key on Apple II-family computers that generates the Escape character. The Escape key is labeled *Esc*. In many applications, pressing *Esc* allows you to return to a previous menu or to stop a procedure.

even parity: In data transmission, the use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an even number, used as a means of error checking. Compare **MARK parity**, **odd parity**.

expansion slot: A socket into which you can install a peripheral card. Sometimes called a *peripheral slot*. See also **auxiliary slot**.

Extended 80-Column Text Card: See **Apple IIe Extended 80-Column Text Card**.

firmware: Programs stored permanently in read-only memory (ROM). Such programs (for example, the Applesoft Interpreter and the Monitor program) are built into the computer at the factory. They can be executed at any time but cannot be modified or erased from main memory.

frequency: The rate at which a repetitive event recurs. In alternating current (AC) signals, the number of cycles per second. Frequency is usually expressed in **hertz** (cycles per second), **kilohertz**, or **megahertz**.

game I/O connector: A 16-pin connector inside all the open models of the Apple II, originally designed for connecting hand controls to the computer, but also used for connecting some other peripheral devices. Compare **hand control connector**.

GLU: Acronym for *general logic unit*, a class of custom integrated circuits used as interfaces between different parts of the computer.

graph: A pictorial representation of data.

graphics: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text**.

hand controls: Peripheral devices, with rotating dials and push buttons. Hand controls are used to control game-playing programs, but they can also be used in other applications.

hand control connector: A 9-pin connector on the back panel of the Apple IIe, Apple IIc, and Apple IIGS computers, used for connecting hand controls to the computer. Compare **game I/O connector**.

handshaking: The exchange of status information between a DCE and a DTE used to control the transfer of data between them. The status information can be the state of a signal connecting the DCE and the DTE, or it can be in the form of a character transmitted with the rest of the data. See **Data Set Ready**, **Data Terminal Ready**, **Data Carrier Detect**, **XON**, **XOFF**.

hertz: The unit of frequency of vibration or oscillation, defined as the number of *cycles per second*. Named for the physicist Heinrich Hertz and abbreviated *Hz*. See **kilohertz**, **megahertz**.

hexadecimal: The base-16 system of numbers, using the ten digits 0 through 9 and the six letters A through F. Hexadecimal numbers can be converted easily and directly to binary form, because each hexadecimal digit corresponds to a sequence of four bits. Hexadecimal numbers are usually preceded by a dollar sign (\$).

high-order byte: The more significant half of a memory address or other multi-byte quantity. In the 6502 microprocessor used in the Apple II family of computers, the **low-order byte** of an address is usually stored first, and the high-order byte second. (In the 68000 microprocessors used in the Macintosh family, the high-order byte is stored first.)

Hi-Res: A high-resolution display mode on the Apple II family of computers, consisting of an array of points, 280 wide by 192 high, with 6 colors.

Hz: See **hertz**.

128K Apple II: Any standard Apple II with both main and auxiliary 64K banks of RAM. That includes all models of the Apple IIc and some models of the Apple IIe, including those with the Extended 80-Column Text Card installed. The Apple IIGS is not a 128K Apple II in the strict sense, even though it includes both 64K banks of RAM and is capable of running programs designed for a 128K Apple II.

IC: See **integrated circuit**.

icon: An image that graphically represents an object, a concept, or a message.

index register: A register in a computer processor that holds an index for use in indexed addressing. The 6502 and 65C816 microprocessors used in the Apple II family of computers have two index registers, called the **X register** and the **Y register**.

indexed addressing: A method used in machine-language programming to specify memory addresses. See also **memory location**.

input: (n) Information transferred into a computer from some external source, such as the keyboard; a disk drive, or a modem.

input/output (I/O): The process by which information is transferred between the computer's memory and its keyboard or peripheral devices.

instruction: A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

integrated circuit: An electronic circuit—including components and interconnections—entirely contained in a single piece of semiconducting material, usually silicon. Often referred to as an *IC* or a *chip*.

interactive: Operating by means of a dialog between the computer system and a human user.

interface: (1) The point at which independent systems or diverse groups interact. The devices, rules, or conventions by which one component of a system communicates with another. Also, the point of communication between a person and a computer. (2) The part of a program that defines constants, variables, data structures, and procedure-calling conventions, rather than procedures themselves.

interface card: A peripheral card that implements a particular interface (such as a parallel or serial interface) by which the computer can communicate with a peripheral device such as a printer or modem.

interrupt: A temporary suspension in the execution of a program that allows the computer to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

I/O: See **input/output**.

I/O device: Input/output device. A device that transfers information into or out of a computer.

I/O link: A fixed location that contains the address of an input/output subroutine in the computer's Monitor program.

IWM: Abbreviation for *Integrated Woz Machine*, the custom chip used in built-in disk ports on Apple computers.

joystick: A peripheral device with a lever, typically used to move creatures and objects in game programs; a joystick can also be used in applications such as computer-aided design and graphics programs.

K: See **kilobyte**.

keyboard: The set of keys, similar to a typewriter keyboard, used for entering information into the computer.

kilobit: A unit of measurement, 1024 bits, commonly used in specifying the capacity of memory ICs. Not to be confused with **kilobyte**.

kilobyte (K): A unit of measurement of computer data or memory, consisting of 1024 (2^{10}) bytes. When used this way, *kilo* (from the Greek, meaning a thousand) stands for 1024. Thus, 64K memory equals 65,536 bytes. See also **megabyte**.

kilohertz: A unit of measurement of frequency, equal to 1000 hertz (abbreviated kHz). See also **megahertz**.

KSW: The symbolic name of the location in the computer's memory where the standard input link (namely, to the keyboard) is stored. *KSW* stands for *keyboard switch*.

language card: A peripheral card that, when installed in slot 0 of a 48K Apple II or Apple II Plus, gives the computer a total of 64K of memory. In Apple II's with 64K or more of memory, the part of memory equivalent to that occupied by a language card is sometimes called language-card memory.

line length: The number of characters that fit in a line on the screen or on a page.

location: See **memory location**.

logic board: See **main logic board**.

low-order byte: The least significant byte of a memory address or other multi-byte quantity. In the 6502 and 65C816 microprocessors used in the Apple II family of computers, the low-order byte of an address is usually stored first, and the high-order byte last. (In the 68000 microprocessors used in the Macintosh family, the high-order byte is stored first.)

Lo-Res: The lowest-resolution graphics mode on the Apple II family of computers, consisting of an array of blocks 48 high by 40 wide with 16 colors.

m flag: One of three flag bits in the 65C816 processor that programs use to control the processor's operating modes. In native mode, the setting of the m flag determines whether the accumulator is 8-bits wide or 16-bits wide. See **e flag**, **x flag**.

Macintosh: A family of Apple computers built around 68000 microprocessors, having high-resolution black-and-white displays and using mouse devices for choosing commands and for drawing pictures.

main logic board: A large circuit board that holds RAM, ROM, the microprocessor, custom-integrated circuits, and other components that make the computer a computer.

main memory: The part of a computer's memory whose contents are directly accessible to the microprocessor; usually synonymous with **random-access memory (RAM)**.

MARK parity: A bit of value 1 appended to a binary number for transmission. The receiving device checks for errors by looking for this value on each character. Compare **even parity**, **odd parity**.

Mega II: A custom large-scale integrated circuit that incorporates most of the timing and control circuits of the standard Apple II. It addresses 128K of RAM organized as 64K main and auxiliary banks and provides the standard Apple II video display modes, both text (40-column and 80-column) and graphics (Lo-Res, Hi-Res, and Double Hi-Res).

megabit: A unit of measurement, 1,048,576 (2^{16}) bits or 1024 kilobits, commonly used in specifying the capacity of memory ICs. Not to be confused with **megabyte**.

megabyte: A unit of measurement of computer data or memory, equal to 1,048,576 bytes or 1024 kilobytes; abbreviated Mb.

megahertz: A unit of measurement of frequency, equal to 1,000,000 hertz (abbreviated MHz). See also kilohertz.

memory: The hardware component of a computer system that stores information for later retrieval. See also main memory, random-access memory, read-only memory, read-write memory.

memory location: A unit of main memory that is identified by an address and can hold a single item of information of a fixed size. In the Apple II family of computers, a memory location holds one byte.

memory-mapped I/O: The method used for I/O operations in Apple II computers, where certain memory locations are attached to I/O devices, and I/O operations are just memory load and store instructions.

MHz: Abbreviation for megahertz, one million hertz. See hertz.

microprocessor: A computer processor contained in a single integrated circuit. The microprocessor is the central processing unit (CPU) of the microcomputer. Examples include the 6502 and 65C816 microprocessors used in the Apple II family of computers and the 68000 microprocessor used in the Macintosh family.

microsecond: One millionth of a second. Abbreviated μs .

millisecond: One thousandth of a second. Abbreviated *ms*.

mode: A state of a computer or system that determines its behavior. A manner of operating.

modem: Short for *MODulator/DEModulator*. A peripheral device that links a computer to other computers and information services using the telephone lines.

monitor: See video monitor.

MOS: Abbreviation for *metal oxide silicon*, a method of semiconductor integrated-circuit fabrication on silicon using layers of silicon dioxide in the make-up of the devices. Compare CMOS.

mouse: A small device you move around on a flat surface next to your computer. The mouse controls a pointer on the screen whose movements correspond to those of the mouse. You use the pointer to select operations, to move data, and to draw with in graphics programs.

mouse button: The button on the top of the mouse. In general, pressing the mouse button initiates some action on whatever is under the pointer, and releasing the button confirms the action.

NTSC: (1) Abbreviation for *National Television Standards Committee*. The committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC, also called **composite**, because it combines all the video information, including color, into a single signal.

odd parity: In data transmission, the use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an odd number; used as a means of error checking. Compare even parity, MARK parity.

128K Apple II: Any standard Apple II with both main and auxiliary 64K banks of RAM. That includes all models of the Apple IIc and some models of the Apple IIe, including those with the Extended 80-Column Text Card installed. The Apple IIGS is not a 128K Apple II in the strict sense, even though it includes both 64K banks of RAM and is capable of running programs designed for a 128K Apple II.

opcode: See operation code.

Open Apple: A modifier key on some Apple II-family keyboards; on the Apple IIGS keyboard, the equivalent key is marked with both an Apple icon and a spinner, the icon used on some Macintosh keyboards, and called simply the *Apple key*.

operation code: The machine-language representation of a computer instruction.

overrun: A condition that occurs when the processor does not retrieve a received character from the receive data register of a communications interface device before the subsequent character arrives to occupy that register.

page: (1) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256. (2) An area of main memory containing text or graphical information being displayed on the screen.

page zero: See zero page.

parallel interface: An interface in which several bits of information (typically 8 bits, or 1 byte) are transmitted simultaneously over different wires or channels. Compare serial interface.

parity: Sameness of level or count, usually the count of 1 bits in each character, used for error checking in data transmission. See even parity, MARK parity, odd parity, parity bit.

parity bit: A bit used to check for errors during data transmission. Depending on the number of 1 bits in a transmission, the parity bit is set to 1 or 0 to make the total number of 1 bits even or odd.

peripheral: (adj) At or outside the boundaries of the computer itself, either physically (as a peripheral device) or in a logical sense (as a peripheral card). (n) Short for *peripheral device*.

peripheral card: A removable printed-circuit board that plugs into one of the computer's expansion slots. Peripheral cards enable the computer to use peripheral devices or to perform other subsidiary or peripheral functions.

peripheral device: A piece of hardware—such as a video monitor, disk drive, printer, or modem—used in conjunction with a computer and under the computer's control. Peripheral devices are often (but not necessarily) physically separate from the computer and connected to it by wires, cables, or some other form of interface. They often require **peripheral cards**.

peripheral slot: See **expansion slot**.

phase: (1) A stage in a periodic process. A point in a cycle. For example, the 65C816 microprocessor uses a clock cycle consisting of two phases called $\Phi 0$ and $\Phi 1$. (2) The relationship between two periodic signals or processes.

pixel: Short for *picture element*. The smallest dot you can draw on the screen. Also, a location in video memory that corresponds to a point on the graphics screen when the viewing window includes that location. In the Macintosh display, each pixel can be either black or white, so it can be represented by a bit; thus, the display is said to be a **bitmap**. In the Super Hi-Res display on the Apple IIGS, each pixel is represented by either two or four bits; the display is not a **bitmap**, but rather a **pixelmap**.

pixelmap: A set of values that represents the positions and states of the set of pixels making up an image. Compare **bitmap**.

port: A socket on the back panel of the computer where you can plug in a cable to connect a peripheral device, another computer, or a network.

processor: The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory. See **microprocessor**.

programmable read-only memory: A type of ROM device that is programmed after fabrication, unlike ordinary ROM devices, which are programmed during fabrication.

PROM: See **Programmable Read-Only Memory**.

prompt character: A text character displayed on the screen, usually just to the left of a **cursor**, where your next action is expected. The prompt character often identifies the program or component of the system that's prompting you. For example, Applesoft BASIC uses a square bracket prompt character (`[]`); the system Monitor program, an asterisk (`*`); and the Mini-assembler, an exclamation point (`!`).

QWERTY keyboard: The standard layout of keys on a typewriter keyboard; its name is formed from the first six letters on the top row of letter keys. Compare **Dvorak keyboard**.

RAM Disk: A feature of some operating systems making it possible to use programmable memory (RAM) as a disk volume. Large applications designed for machines with limited amounts of RAM must load program segments from disk as needed; on machines with RAM Disk, the entire application is first loaded into RAM, where it runs as if still resident on disk, only much faster.

random-access memory (RAM): Memory in which information can be referred to in an arbitrary or random order. As commonly used, RAM means the part of memory available for programs from a disk; the programs and other data are lost when the computer is turned off. (Technically, the read-only memory (ROM) is also *random access*, and what's called RAM should correctly be termed *read-write memory*.) Compare *read-only memory*, *read-write memory*.

read-only memory (ROM): Memory whose contents can be read, but not changed; used for storing *firmware*. Information is placed into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off. Compare *random-access memory*, *read-write memory*, *write-only memory*.

read-write memory: Memory whose contents can be both read and changed (or written to); commonly called RAM. The information contained in read-write memory is erased when the computer's power is turned off and is permanently lost unless it has been saved on a disk or other storage device. Compare *random-access memory*, *read-only memory*.

register: A location in a processor or other device where an item of information is held and modified under program control.

RGB: Abbreviation for *red-green-blue*, a method of displaying color video by transmitting the three primary colors as three separate signals. There are two ways of using RGB with computers: **TTL RGB**, which allows the color signals to take on only a few discrete values; and **analog RGB**, which allows the color signals to take on any values between their upper and lower limits, for a wide range of colors.

RGB monitor: A type of color monitor that receives separate signals for each color (red, green, and blue). See *composite video*.

ROM: See *read-only memory*.

ROM Disk: A feature of some operating systems making it possible to use read-only memory (ROM) as a disk volume. Often used for making applications permanently resident. See also **RAM Disk**.

row: A horizontal line of character cells or graphics pixels on the screen.

RS-232: A common standard for serial data-communication interfaces.

RS-422: A standard for serial data-communication interfaces, different from the RS-232 standard in its electrical characteristics and in its use of differential pairs for data signals. The serial ports on the Apple IIGS use RS-422 devices modified so as to be compatible with RS-232 devices.

SCC: Acronym for *Serial Communications Controller*, a type of communications IC used in the Apple IIGS computer. See **ACIA**.

screen holes: Locations in the text display buffer (text Page 1) used for temporary storage either by I/O routines running in peripheral-card ROM or by firmware routines addressed as if they were in card ROM. Text Page 1 occupies memory from \$400 to \$7FF; the screen holes are locations in that area that are neither displayed nor modified by the display firmware.

serial interface: An interface in which information is transmitted sequentially, a bit at a time, over a single wire or channel. Compare **parallel interface**.

serial port: The connector for a peripheral device that uses a **serial interface**.

silicon: A solid, crystalline chemical element (symbol Si) from which integrated circuits are made. Silicon is a *semiconductor*; that is, it conducts electricity better than insulators, but not as well as metallic conductors. Silicon should not be confused with silica—that is, silicon dioxide, such as quartz, opal, or sand—or with silicone, any of a group of organic compounds containing silicon.

Simplified Keyboard: See **Dvorak keyboard**.

64K Apple II: Any standard Apple II that has at least 64K of RAM. That includes the Apple IIc, the Apple IIe, and an Apple II or Apple II Plus with 48K of RAM and the Language Card installed.

6502: The microprocessor used in the Apple II, in the Apple II Plus, and in early models of the Apple IIe. The 6502 is an MOS device with 8-bit data registers and 16-bit address registers.

65C02: A CMOS version of the 6502; the microprocessor used in the Apple IIc and in the enhanced Apple IIe.

65C816: The microprocessor used in the Apple IIGS. The 65C816 is a CMOS device with 16-bit data registers and 24-bit address registers.

68000: The microprocessor used in the Macintosh and Macintosh Plus. The 68000 has 32-bit data and address registers.

slot: A narrow socket inside the computer where you can install peripheral cards. Also called an **expansion slot**.

soft switch: A location in memory that produces some specific effect whenever its contents are read or written.

stack: A list in which entries are added (pushed) or removed (popped) at one end only (the top of the stack), causing them to be removed in last-in, first-out (LIFO) order. Compare **queue**.

Standard Apple II: Any computer in the Apple II family except the Apple IIGS. That includes the Apple II, the Apple II Plus, the Apple IIe, and the Apple IIc.

Standard Apple Numerics Environment: Apple's implementation of IEEE standard floating-point arithmetic, used on the Apple II and Macintosh families of computers.

start bit: One or two bits that indicate the beginning of a character in a string of serially transmitted characters.

stop bit: A bit indicating the end of a character in a string of serially transmitted characters.

strobe: A signal whose change is used to trigger some action.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a computer system comprising a processor, keyboard, monitor, and disk drive.

system configuration: See **configuration**.

text: (1) Information presented in the form of readable characters. (2) The display of characters on a display screen. Compare **graphics**.

transistor-transistor logic (TTL): (1) A family of integrated circuits having bipolar circuit logic; TTL ICs are used in computers and related devices. (2) A standard for interconnecting such circuits, which defines the voltages used to represent logical zeros and ones.

TTL: See **transistor-transistor logic**.

TTL RGB: A type of video monitor that can accept only a limited number of digital values and display only a correspondingly limited number of colors. Compare **analog RGB**.

type-ahead buffer: A buffer that accepts and holds characters that are typed faster than the computer can process them.

VBL: Short for *vertical blanking*, an interrupt signal generated by the video timing circuit each time it finishes a vertical scan, 60 times a second.

vector: (1) The starting address of a program segment when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

video: (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form.

video monitor: A display device that can receive video signals by direct connection only, and that cannot receive broadcast signals such as commercial television. Can be connected directly to the computer as a display device.

word: A group of bits that is treated as a unit. The number of bits in a word is a characteristic of each particular computer; in the Apple IIGS, words are sixteen bits wide.

wraparound: The automatic continuation of text from the end of one line to the beginning of the next; wraparound means that you don't have to press the Return key at the end of each line as you type.

write-only memory: A form of computer memory into which information can be stored but never, ever retrieved. For more information, refer to *The Life of Homberg T. Farnsfarfle*, by Bruce Tognazzini.

x flag: One of three flag bits in the 65C816 processor that programs use to control the processor's operating modes. In native mode, the setting of the x flag determines whether the index registers are 8-bits wide or 16-bits wide. See **e flag**, **m flag**.

XON: A special character (ASCII value \$13) used for controlling the transfer of data between a DTE and a DCE. See **handshaking**, **XOFF**.

XOFF: A special character (ASCII value \$11) used for controlling the transfer of data between a DTE and a DCE. When one piece of equipment receives an XOFF character from the other, it stops transmitting characters until it receives an XON. See **handshaking**, **XON**.

X register: One of the two index registers in the 6502 microprocessor.

Y register: One of the two index registers in the 6502 microprocessor.

zero page: The first page (256 bytes) of memory in the Apple II family of computers, also called *page zero*. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory. The 65C816 microprocessor used in the Apple IIGS has a relocatable zero page called the **direct page**.

