Understanding the Apple IIe

by Jim Sather

Foreword by Steve Wozniak

QUALITY SOFTWARE
Computer Book Division

Understanding the Apple IIe

by James Fielding Sather

BRADY COMMUNICATIONS, CO., INC.
A Prentice-Hall Publishing Company
BOWIE, MARYLAND 20715

Quality Software
21601 Marilla Street
Chatsworth, CA 91311

Production Editor: Kathryn M. Schmidt
Editorial Assistant: Tom Weinstein
Original Schematics and Diagrams: James Fielding Sather
Art Director and Cover Design: Vic Grenrock
Cover Art: George Garcia
Schematic Art and Compositor: Ron Widman
Photography: Gainsforth Studios
Printed By: Griffin Printing

ISBN 0-8359-8019-7

87 86 85 5 4 3 2 1

Printed in the United States of America

foreword

I will never forget the first conversation I had with Jim Sather. He was then in the process of writing his first book, **Understanding the Apple II.** *Jim and I discussed the details, anomalies, oversights, and paradoxes of the Apple II hardware as we drove the LA freeways. Designers like myself find it very rewarding to encounter others who understand and appreciate what we feel are the tricks and magic of our circuits. I was able to add to the magic by explaining the unusual framework in which the computer was designed. From my conversations with him, and from his writing, it is obvious that Jim has a contagious enthusiasm about Apple computers, and this enthusiasm is sure to spread to readers of his books. In* **Understanding the Apple II,** *Jim provided the definitive treatment of Apple II hardware. He has now followed that effort with the equally definitive* **Understanding the Apple IIe.**

Patterned after the earlier book, **Understanding the Apple IIe** *leaves no stone unturned in the search into the inner workings of the Apple IIe computer. All facets of the Apple are revealed, from basic microprocessor operation to the inner secrets of the Apple IIe custom ICs. Disk controller operation—my favorite subject—is explained in great detail. Numerous programming examples illustrate the application of hardware knowledge.*

Anyone who is at all concerned about the workings of the Apple IIe will benefit from this book, as will students and programmers who have a need for reliable hardware reference material. It is an inclusive source for a great variety of Apple IIe information. The Apple IIe holds a special niche in the history of microcomputers. Documentation of this quality is worthy of the computer it describes.

preface

It has been close to a decade since an unknown kid, having some fun in his own creative way, built the first Apple computer. What a difference a decade makes. Our boy is well known now, and he and his pals have built millions of Apples with which millions of people have had a tremendous amount of fun in their own creative ways.

The creative ways of different people lead them in different directions, and not all Apple owners use their computer for the same purposes. Yet diverse as they are, people who use their Apple have a common need for knowledge and understanding of the workings of the computer. Most of them will teach themselves almost everything they learn about the Apple, so they also have a need for tutorial literature and meaningful reference material to guide them down their chosen paths.

The purpose of *Understanding the Apple IIe* is to provide tutorial descriptions and reference material concerning the most basic of Apple IIe related knowledge. It contains explanations of how the hardware works and how programmers make the hardware work. Emphasis is placed on assisting the reader in attaining operational knowledge of the Apple IIe. Operational knowledge consists of knowing what the Apple IIe can do, knowing how to make it do it, and knowing what a controlling program is making the Apple IIe do. By way of assisting the reader in achieving his goals, the goals of this book are:

1. To provide clear descriptions of microcomputer fundamentals and of the operational features of the Apple IIe.
2. To provide examples that show how knowledge of the operational features of the Apple IIe can be applied.
3. To provide meaningful reference material concerning Apple IIe hardware and operational features.
4. To serve as a textbook for Apple-based high school or university courses teaching computer fundamentals.
5. To fill information gaps in Apple IIe literature by describing previously undocumented operational features.

Those who will benefit from reading *Understanding the Apple IIe* are inquiring people who want to spend some time learning about this machine. Generally speaking, this refers to those persons who program the Apple IIe in any language. It is recognized that different people will carry their investigation to different depths. For those who do not have the time or desire to reach the greater depths, the overview, bus structure, and I/O chapters (Chapters 1, 2, and 7), as well as the application notes at the end of every chapter, are recommended as providing a good foundation for understanding the Apple IIe. As a textbook for students or a learning guide to hard core enthusiasts, cover to cover reading is recommended.

While an inquiring mind is the only qualification required of a reader of this book, certain sections will be difficult for those readers without some background knowledge. In order of descending importance, helpful background knowledge includes understanding of BASIC programming language, hexadecimal and binary number systems, 6502 assembly language, and technical illustrative aids such as timing diagrams, truth tables, and schematic diagrams. It should be noted by all readers that (except for the technical aids) they will eventually have to acquire the listed background knowledge if they are to achieve a real understanding of the Apple IIe computer. It is hoped that the nontechnical aids and language in *Understanding the Apple IIe* are sufficiently descriptive, and that a technical background, although helpful, is not necessary. In general, the later chapters contain more detailed and technical information than the earlier chapters, and the earlier sections in each chapter are less technically oriented. Appendices E and F contain some basic information on number systems and circuit symbols for those readers who come to this book with no previous knowledge of these subjects.

Even though *Understanding the Apple IIe* is not a programming instruction manual, many programming examples that illustrate applications of the principles being discussed are given in the body of the text. Where possible, these examples are written

in BASIC so that the clearest attainable level of illustration results. In addition, a number of software application notes are included at the end of various chapters which further demonstrate the application of principles. These programming notes are included because understanding the Apple IIe includes a combination of programming knowledge and hardware knowledge. Unless noted otherwise, all software examples are creations of the author and are hereby placed in the public domain. The author requests that he be given credit as the programmer in all reproductions of these programs.

A number of hardware application notes are also included at the ends of chapters. Some of these notes describe hardware projects which demonstrate relevant principles. Other notes are simple descriptions of hardware modifications that enhance operation in some way. Figure 4.7 is an original design of the author. Readers are encouraged to study, build, or integrate it into their own designs. The author requests that he be given credit as the designer in any reproduction or other use of this schematic. The D MAnual Controller is being manufactured by the Southern California Research Group, and is available by mail as noted in Chapter 4.

Several hardware application notes detail modifications to the Apple or Apple peripherals. Please read the NOTE OF CAUTION following the Table of Contents before performing any modifications to your equipment. It is recommended that readers unskilled in electronics workmanship who desire a modification have the work performed at a computer dealership or by a skilled friend. Persons who modify their hardware should be able, or know someone who is willing and able, to repair the modified assembly if it should fail.

Understanding the Apple IIe is the companion of my previously published work, *Understanding the Apple II**. These two books are identical in format and outline, one describing the Apple II computer and the other describing the Apple IIe. Readers of both books will find that, where operational features in the two computers are identical, the text in the two books is identical. Those readers will also find that some application notes which are relevant to both the Apple II and Apple IIe are found in both books. To the extent that operational features and hardware implementation in the Apple IIe is different than that of the Apple II, *Understanding the Apple IIe* is different from *Understanding the Apple II*.

*Quality Software, 1983.

In deference to readers who have experience only with the Apple IIe, descriptions in *Understanding the Apple IIe* assume that the reader is not familiar with the functioning of the older Apple II. However, Apple II features and functions are sometimes described in order to clarify differences between the two computers or to explain why Apple IIe features exist. Some notes on differences between the Apple II and IIe are contained in Appendix I.

There are differences among Apples that are sold in various regions of the world, and it is sometimes difficult to make statements that are accurate for all versions. Generally, descriptions in this book pertain to the Apple IIe as it is sold in the USA with separate sections devoted to descriptions of export versions. Readers in other countries should be aware that some descriptions, in particular those dealing with signal frequency and video generation, may give details that are not accurate in their country. Those readers should rely on the sections of Chapters 3 and 8 that deal directly with international Apples for guidance. Additionally, it should be noted that program listings in Figures 3.11 and 3.12 have to be modified if they are to operate correctly in 50 Hz display scanning Apples such as those found in Europe.

Figures 1.1, 3.8, 3.10, 5.3, 5.13, 7.1, and 8.5 illustrate functions internal to the Apple IIe special purpose integrated circuits (the IOU, MMU, and timing HAL). These drawings are my own representations of those internal functions, based on my observations of Apple IIe signals and features. These drawings do not accurately show internal circuit detail, but are intended only to accurately depict internal circuit functions.

Understanding the Apple IIe is the result of an intensive investigation of the Apple IIe computer by the author. There is no other source of much of the information covered here, and the possibility of error exists on the part of the author. For those errors which do exist, the author is truly sorry.

The Apple IIe is not a perfect computer, Apple Computer, Inc. is not a perfect company, and I am not a perfect author. There are many opinions of the author in the body of the text, and some of them are negative toward the Apple IIe or the company that manufactures it. The reader must rely on his own judgment to evaluate these opinions. Although I am sometimes critical of Apple Computer, Inc., I acknowledge that the actions of this company have enriched my life. Although I am sometimes critical of the Apple IIe, I believe it is the best personal computer that money can buy.

Contents

Contents

Dedication

On behalf of my brothers and sisters,
Lee, Jenny, Tim, Mary, Mike, and Joe,

to my father,
Fredrick Ingwald Sather,
with love and respect.

Acknowledgements

My wife, Deborah, still tolerates me and proofreads all text before I submit it.

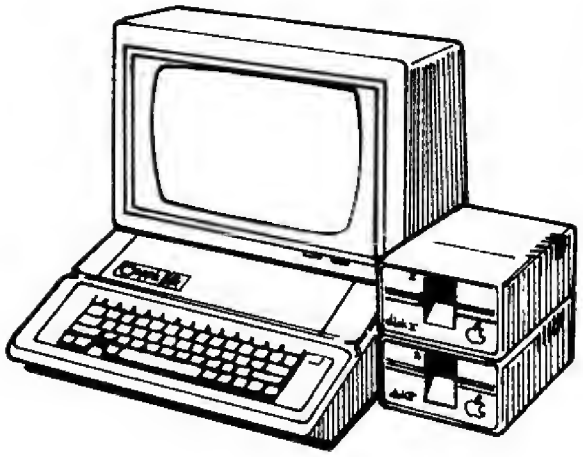
When I couldn't answer my own questions, I asked the telephone. Thanks for information, guidance, and answers to: Paul Darcy (PAL motherboard, history, general), Peter Baum (much general information and assistance), Pieter Lechner and Bob Sander-Cederlof (ProDOS), Walt Broedner (history), Dan Fischer (interrupts), Jeff Mazur (general), and Eric Waller, Roger Wilbur, and Mike England (maintenance procedures).

Every page of this book gives evidence of effort put forth by the people at Quality Software. What you see is far more polished than what I gave them.

Note of Caution

Several of the Application Notes in Understanding the Apple IIe contain procedures for modifying the Apple IIe computer and peripheral cards. Modification of your Apple or peripherals may void your warranty if the warranty period has not yet lapsed. It may also increase your out of warranty repair costs should the modified unit fail in the future. The decision to perform any of the modifications described in Understanding the Apple IIe rests solely with the owner of the hardware concerned. Neither Quality Software nor the author bears responsibility for any negative consequence of the owner's decision to perform such modifications.

chapter 1



The Apple IIe— An Overview

The following overview is a brief statement of the hardware features of the Apple IIe computer. It is not meant to be a description of everything programs can make the Apple IIe do. Rather, it is a description of the basic capabilities with which computer programmers and peripheral designers work. An attempt is made to explain the technical terms that are used, but newcomers to microcomputers should not be discouraged if some points are not absolutely clear to them. The chapters that follow expand on all topics covered here, and Chapter 2 in particular contains information which will clarify much of Chapter 1.

First and foremost, the Apple IIe is a revised and improved version of the Apple II computer that was designed by Steve Wozniak in the mid-seventies. It is operationally compatible with a 48K Apple II that has a 16K expansion RAM card in Slot 0 and an 80-column text card in Slot 3. The Apple IIe also supports 64K of auxiliary RAM and has an improved keyboard, improved graphics capability, and numerous minor operational improvements, but compatibility with the Apple II is its predominant feature.

Apple's motivation in refining the Apple II was reducing manufacturing costs and eliminating some critical text handling weaknesses of the Apple II. They achieved these goals very nicely and produced a computer that is better than the Apple II but which inherited its personality and many features from the Apple II. The computer that is described here is the Apple IIe, but much of what is said is also true of the Apple II.

APPLE IIe OVERVIEW

The Apple IIe is made up of five physical units: the baseplate and case, the keyboard, the power supply, the speaker, and the motherboard. The speaker, power supply and keyboard are all utility units which plug into the motherboard. It is the motherboard which contains all the uniqueness of the Apple IIe. The motherboard is the Apple IIe, and the Apple IIe is consequently referred to as a single board computer. On one board, it has a microprocessor, memory, video text and graphics output circuitry, seven peripheral expansion slots, an auxiliary expansion slot, and circuitry for communications

with a variety of external devices. These features are part of an organized structure centered around the microprocessor.

The Microprocessor and Bus Structure

The brains of the Apple IIe is a 6502 **microprocessor**. A microprocessor, or **MPU** (MicroProcessing Unit), is a single chip logic device capable of executing stored sequential programs.* A microcomputer is a computer which uses an MPU as its fundamental logic processor.

Digital computers operate to a synchronizing beat known as a **clockpulse**, similar to the beat of music, but over ten thousand times as fast. The 6502 operates to a beat which occurs approximately 1,020,500 times a second. We say that the clockpulse frequency is 1.0205 MegaHertz (MHz) meaning 1.0205 million cycles per second. Actually, there is a clockpulse jitter, which is described in the timing section of Chapter 3. Until we get to that point, just say that the 6502 operates at about 1 MHz. This, incidentally, is slow by modern microprocessor standards. There are 4 MHz 6502 MPUs available now, and other MPUs have faster clockpulse rates than that. With a given MPU, the faster the clock, the faster the execution speed.

The structure of the Apple IIe is that of multiple devices which can communicate with the MPU. Once every clockpulse, the MPU outputs the address of the location which is being communicated with, and it transmits data to or receives data from that location. The address which the MPU is putting out is distributed to all addressable devices in the Apple IIe via the **address bus**, and data is transferred between the MPU and the addressed location via the **data bus**. Associated and distributed with the address bus is the **read/write control** output of the MPU. Read/write control tells the addressed location whether data will be read from it or written to it.

The 6502 has **16 address outputs**, each connected to one line (electrical conductor) of the address bus.** It controls the 16 address lines and the read/write line together by placing a high or a low voltage on each line. The simultaneous condition of the 16 address lines is the 6502 address. The 6502 address is a number between $0 and $FFFF (65535), and the 6502 can access any one of the $10000 (65536) addressed locations in that range.

The 6502 has **eight data input/output lines**, each connected to one line of the data bus. It controls the eight lines when writing and monitors the eight lines when reading, and the simultaneous condition of the eight lines is the 6502 data word. Like the address lines, each of the data lines is brought to a high or a low voltage when information is passed. Each line can be one of two states (high or low), so the information is said to be **two state**, or **binary**. Other common ways of referring to the two states of binary information are true/false, one/zero, and on/off.

A unit of binary information is a **bit**. Whether a line is high or low at a given instant is a bit of information. The 6502 reads or writes and manipulates information eight bits at a time and is therefore classified as an 8-bit MPU. A group of eight bits is a **byte**. The 6502 manipulates and transfers data, one byte at a time, to an addressed location in the Apple IIe bus system.

Most locations which the MPU addresses are **memory** locations. Memory contains the stored program which the MPU is executing and about half of the MPU's time is spent fetching that program. The program is stored sequentially, so fetching the program by the MPU simply involves incrementing the address output while reading the data input and interpreting it as a sequential program. When not fetching the program, the MPU is executing it. This execution involves logical manipulation of data, storage of data at or loading of data from addressed locations determined by the program, changing the program fetching location to somewhere other than the next sequential address, or any combination of these and other functions.

Not all locations addressed by the MPU are memory locations. Program instructions fetched from memory may cause the MPU to address non-memory locations such as the speaker or keyboard. A memory location responds to a read at its address by placing data on the data bus. The speaker responds to a read or a write at its address with sound. The MPU thus controls the speaker via the address bus in an **address decoding** process.

*A chip is another name for an integrated circuit, or IC. It is a unit with a small body and a number of metal pins or leads, and it contains complex electronic circuitry inside. If you look inside the Apple IIe, you will see many little black chips plugged into sockets on or soldered directly to the motherboard. There are four chips that are bigger than all the others, and the 6502 MPU is one of the four big chips.

**As described in Chapters 2 and 4, the 6502 is not connected directly to the address bus. It is connected to the address bus through isolating devices which give the Apple IIe a DMA (Direct Memory Access) capability and allow the 6502 to address the large number of electronic devices connected to the address bus of the Apple IIe.

Address decoding is the only way a 6502 can control other devices, so all programmed control of Apple IIe devices is via address decoding.

Memory

General purpose microcomputers require two types of memory, memory you can change (**RAM**) and memory you can't change (**ROM**).* RAM is necessary so you can store general programs and data. ROM is necessary so the computer has a program to run when it is first turned on.

Both ROM and RAM are random address memories, meaning any specific memory location can be accessed at its specific address. Computer memory is like thousands of light bulbs, each of which may or may not be glowing. If the memory is random access, the microprocessor can communicate with any light bulb it chooses by calling its number. It can, for example, check if light bulb number 25,765 is glowing or not. This is analogous to reading from memory. Telling light bulb number 7,682 to not glow is analogous to writing to memory; the MPU is altering the state of light bulb 7,682. RAM and ROM are functionally identical except that ROM is fixed as if it was etched in stone. You can't turn the light bulbs on or off. You can only check to see if they are on or off.

The MPU cannot really tell whether a light bulb is glowing or not, but it can tell whether the voltage on a line is high or low. RAM is capable of storing the high/low state of its data input when the MPU writes data to a RAM address. Both RAM and ROM are capable of bringing their data outputs high or low in accordance with stored data when the MPU reads data from a RAM or ROM address. In a **positive logic** system like that of the Apple IIe, storing or reading a high voltage is thought of as storing or saving a "1". Storing or reading a low voltage is thought of as storing or saving a "0".

Since the 6502 is an 8-bit MPU, memory must be organized so that it is accessed eight bits, or one byte, at a time. The Apple IIe motherboard has sockets for 65,536 bytes (524,288 bits) of RAM. This is normally referred to as 64K of RAM, meaning 64 Kilobytes. In addition to this motherboard RAM, motherboard timing and memory management fully support an additional 64K of RAM on a card installed in an **auxiliary slot** that is mounted near the front of the motherboard.

The 64K of motherboard RAM in the Apple IIe is functionally similar to the 64K of RAM in an Apple II with Slot 0 16K expansion RAM card. **Low RAM** is the 48K addressed at $0000-$BFFF, and **high RAM** is 16K addressed at $D000-$FFFF with $D000-$DFFF response switched between two 4K banks. Low RAM is the main body of Apple IIe RAM, and it does not share $0000-$BFFF with other motherboard devices. High RAM is secondary RAM that shares $D000-$FFFF response with motherboard ROM. It is disabled for reading, in favor of motherboard ROM, anytime the RESET key is pressed. Auxiliary card RAM is divided the same way as motherboard RAM, so a 128K Apple IIe is the RAM equivalent of two 48K Apple IIs with two 16K RAM cards.

The Apple IIe uses **dynamic RAM** which must be **refreshed**. Memory refresh must occur on a periodic basis or dynamic RAM will not work. It's like a fire that goes out unless someone is constantly pumping the bellows. Dynamic RAM is nice because it's inexpensive, but it requires a lot of external circuitry to support the refresh requirement. The Apple IIe fully supports 64K of motherboard RAM and 64K of auxiliary card RAM in every way, including refresh.

The Apple IIe motherboard contains 16,128 bytes of system **firmware** (programs and data in ROM). This firmware includes a **system monitor**, **Applesoft BASIC**, some separate keyboard-in / video-out routines referred to as the **80-column firmware**, and some system **diagnostic routines**. The monitor tells the Apple IIe what to do at power-up and contains valuable utilities which make the Apple IIe hardware accessible to its user; Applesoft is the BASIC editor and command interpreter normally used in the Apple IIe; the 80-column firmware is an extension of the monitor written to support the Apple IIe 80-column text display; and the firmware diagnostics provide the Apple IIe with a modest self testing capability.

Peripheral Slots

The Apple IIe peripheral slots are similar to a **card cage**. What is a card cage? A card cage is a very versatile physical package for microcomputers and other electronic circuits. It is a row of slots mounted close together into which printed circuit cards are plugged. Behind the slots are hundreds of wires connecting the slots together in accordance with the design purpose. Card cage architecture is

*ROM stands for Read Only Memory, which is accurate, and RAM stands for Random Access Memory, which is the most famous misnomer in all of computer jargon. Both read only memory and read/write memory in the Apple IIe are random access memory, and this book refers to them by their conventional labels, ROM and RAM.

like a house with an intercom system. Just as communication is possible between various rooms of the house, communication is possible between the various cards plugged into the card cage. Each slot in the card cage is a different station in the intercom system.

In a card cage microcomputer, part of the wiring which interconnects the slots is a multiline address bus and data bus, similar to the buses on the Apple IIe motherboard. A microprocessor board can be plugged into any slot, from where it can control communication in the card cage via the address bus. A very nice modern card cage micro would have a multifunction single board microcomputer in one slot and a variety of devices in the other slots. The Apple IIe is exactly that computer, turned inside out. Instead of mounting the main logic board in the card cage, they mounted the card cage on the main board.

The Apple IIe "card cage" consists of seven **peripheral slots** mounted on the back of the motherboard. The address bus and data bus are connected to all the slots, making them addressable extensions of the Apple's basic communication system.* Each slot has a part of the 6502 address range assigned to it, so programs can make the 6502 access a peripheral slot just as if it were a group of memory locations.

Some important 6502 input control signals are tied to pins on the peripheral slots. They are RESET', READY, NMI' (Non-Maskable Interrupt), and IRQ' (Interrupt ReQuest). These signals are all described in greater detail in the 6502 section of Chapter 4. Their connection to the peripheral slots means that the processor can be interrupted, stopped, started, and reset from any peripheral card. It also means that any peripheral card can be designed to respond to these control signals. For example, pressing RESET at the keyboard resets the 6502 and additionally turns off the floppy disk drive. The disk drive controller is designed to respond to the RESET' signal which is pulled low when RESET is pressed. RESET', incidentally, is read "reset prime." In this book, the prime behind the name of a logic term is used to signify that a signal is active or true when a low voltage is present.** It is an aid to understanding the logic functions of a given signal. Knowing this, you could guess from the second sentence of this paragraph that the 6502 is interrupted and reset by low voltages on the NMI', IRQ', and RESET' lines, and enabled by a high voltage on the READY line.

Another peripheral slot signal which affects the 6502 but isn't connected directly to it is the DMA' signal. **DMA** stands for **Direct Memory Access** and refers to direct memory access from the peripheral slots. The DMA' line does a bit more than give the slots access to memory, however. It allows a card in a slot to isolate the 6502 from the address bus and data bus and take control of communication in the bus system. This means that a peripheral card can control all hardware features of the Apple IIe. It is as if you could plug a Suzy brain into Johnny and have the Suzy brain control Johnny's body, a concept much in vogue in some circles.

There are signals connected to the peripheral slots other than those that have been mentioned. They provide various capabilities so peripherals can be designed to be fully integrated into the Apple structure. These signals include timing and control inputs, power supply voltages, and control signals decoded from address ranges on the address bus. The purposes of these signals will be fully explained in later chapters.

The Auxiliary Slot

The auxiliary slot is a 60-pin slot that is physically separated from the peripheral slots. Like a peripheral slot, the auxiliary slot holds a card that is designed to augment the features of the motherboard. Unlike a peripheral slot, the auxiliary slot does not feature full connection to the address bus and data bus and is not supported as an I/O port by Apple IIe firmware.

Rather than acting as an I/O port, the auxiliary slot is designed to accept cards that interact with the RAM, video generation, and/or timing generation circuitry of the motherboard. It most commonly holds a 64K RAM card that enables video display of 80-columns of text, enables doubling of the Apple IIe video graphics horizontal resolution, and makes a total of 128K of RAM accessible to the Apple IIe MPU. Other functions such as RGB (Read-Green-Blue) video signal generation can also be performed

*As described in Chapters 2 and 7, the peripheral slots are actually connected to the data bus through a bidirectional bus driver that enables the 6502 to communicate with a large number of peripheral card devices via the data bus.

**Most published computer literature will overscore a logic term, rather than placing a prime symbol behind it, to signify that it is active when low. In using the prime notation, *Understanding the Apple IIe* is following the convention used by Apple in the *Apple IIe Reference Manual for IIe Only*. In addition to signifying that a term is active when low, the prime symbol following a logic term can mean that the inversion of that logic term is being referred to. Please see Appendix E for further discussion of this subject.

by auxiliary slot cards, but such alternate function cards will probably always contain at least enough RAM to support the Apple IIe 80-column text display. Additionally, production and service facility auxiliary slot cards can be designed to monitor important Apple IIe timing and video generation signals and inject substitutes for many of those signals to the motherboard.

The MMU, IOU, and Timing HAL

In addition to fundamental building blocks like the MPU, ROM, RAM, and an I/O capability, a microcomputer has a large amount of associated circuitry that supports the operation of the fundamental building blocks. In the Apple IIe, much of this circuitry is concentrated in two custom VLSI (Very Large Scale Integration) ICs, the MMU and the IOU. These custom ICs are very complex integrated circuits, co-designed by Apple and an IC manufacturer* to perform logical functions required in the Apple IIe.

The **MMU** (Memory Management Unit) contains programmable **soft switches** and address decoding circuitry which define the overall memory and I/O configuration of the Apple IIe. By this, it is meant that the MMU controls which device (RAM, ROM, I/O device, or peripheral card) responds to which addresses. This is a complex task in the Apple IIe, because the memory map can be reconfigured so that the same device does not always respond to a given range of addresses.

Programmable soft switches are very important in the operational scheme of the Apple IIe. They are like a mechanical switch, except that they are switched when they are addressed by the MPU, not by the flip of a finger. Programs maintain control of a number of Apple IIe functions by setting and resetting soft switches that are mechanized in the MMU and IOU. As an example, the RAMRD soft switch is a programmable switch in the MMU that, when set, enables MPU reading from much of auxiliary card RAM. It is set when the controlling program causes the MPU to perform write access to $C003 and reset when the program causes the MPU to perform write access to $C002.

*Custom IC design is a cooperative effort by IC and equipment manufacturers. In the case of the IOU and MMU, Apple employee (and former Synertek employee) Walt Broedner designed the IOU and the MMU within the constraints of the Synertek custom IC program. Synertek is the primary MMU and IOU source, and judging by the IOU in my Apple IIe, American Microsystems (AMI) is an alternate source.

The MMU accomplishes its memory management functions by monitoring the address bus and R/W', and responding to certain addresses by setting or resetting its configuration soft switches. Also, for any address on the address bus and any status of the MMU soft switches, the MMU controls which class of motherboard device will respond to an address. The MMU does this by activating or deactivating various data bus management signals. A second function of the MMU is to convert the MPU address from the 16-line address bus format to the 8-line multiplexed format that is required by dynamic RAM. This subject and all subjects related to the MMU are covered in detail in Chapters 2 and 5.

The **IOU** (I/O Unit) contains circuitry primarily related to the various facets of generating the Apple IIe VIDEO signal. This includes the **video scanner**, a counter that scans RAM for video output when the MPU is not accessing RAM. It also includes circuitry to convert video scanner states to a multiplexed RAM address, soft switches by which the display mode of the Apple IIe is established, and circuitry which is actively involved in processing the RAM resident **display map** to generate the VIDEO signal.

In addition to the display related functions mentioned above, several I/O functions of the Apple IIe are implemented in the IOU. These include parts of the **cassette and speaker output** functions, the **annunciator** outputs, the **KEYSTROBE** (keyboard strobe) soft switch, the keyboard auto repeat function, and the capability to transmit the AKD (Any Key Down) line to a line of the data bus so a program can determine when a keyboard key is being held down. The varied IOU tasks span topics covered in several chapters of *Understanding the Apple IIe*. Figure 1.1 is a general diagram of the IOU that shows chapters and figures in which the IOU functions are discussed and illustrated.

A third special purpose VLSI IC on the Apple IIe motherboard is the **timing HAL**. A HAL (Hard Array Logic) is an IC, designed by a manufacturer to perform logic functions within a general format. The specific logic functions that the IC is to perform are specified by the buyer—in this case, Apple Computer, Inc. The timing HAL is similar to a ROM, except that the HAL purchaser specifies logic functions instead of memory contents.

The HAL in the Apple IIe is used in the process of generating the timing signals that synchronize functions throughout the motherboard. The nature of these timing signals and the details of their generation are discussed in Chapter 3.

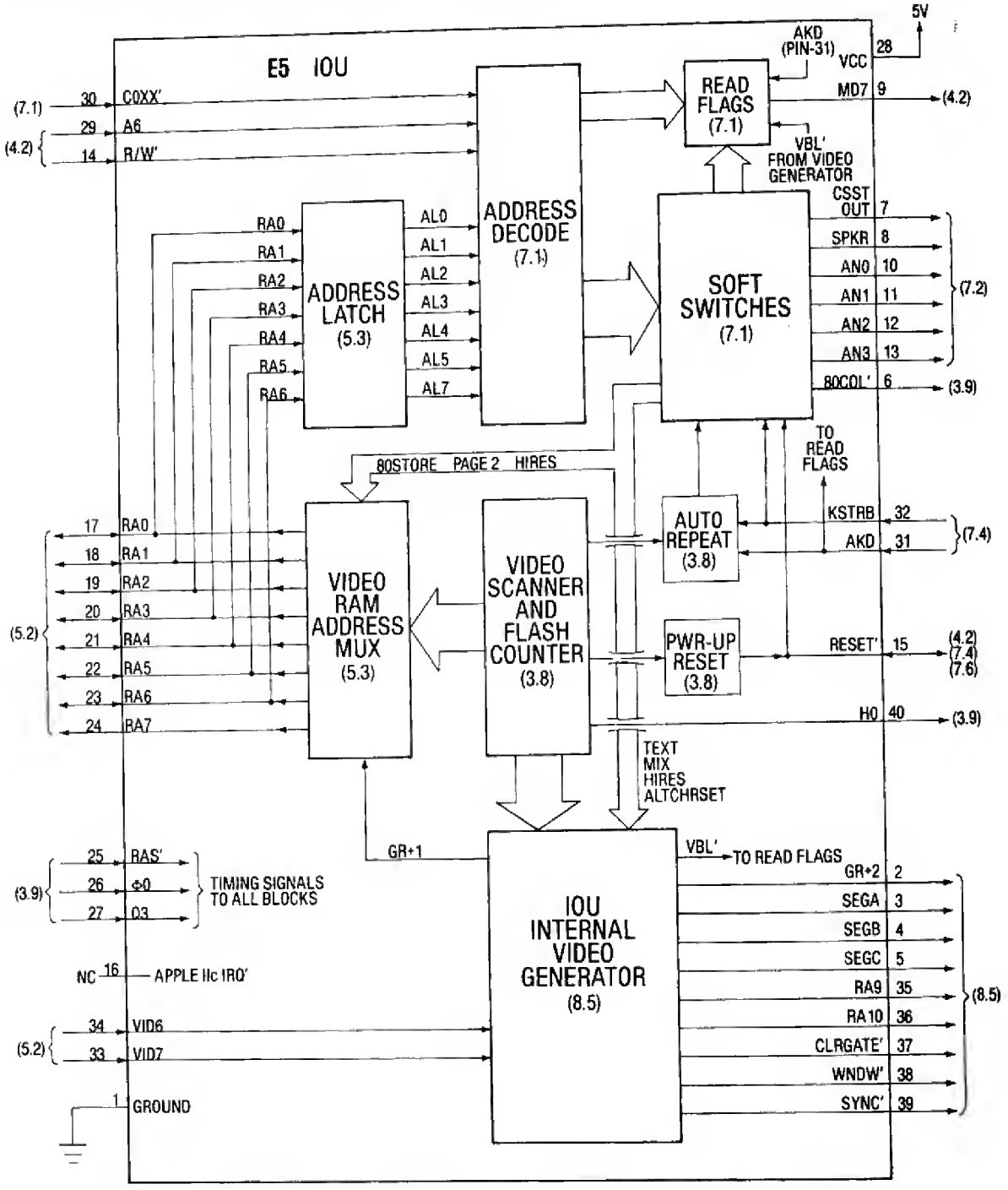


Figure 1.1 IOU Functions and Pin Assignments.

Video Output

The primary output of the Apple IIe is video. This video is color compatible with the television system used in the country in which an Apple IIe is sold. There are two versions of the motherboard—one which outputs video compatible with the NTSC television system used in America and other areas, and one which outputs video compatible with the PAL television system used in most of western Europe and other areas. An Apple IIe in a given country will contain the version of the motherboard compatible with that country's television system. Additionally, the video and keyboard ROMs will be tailored to the requirements of that country's language or languages.

Video from the Apple IIe can be directly input to a color or monochrome video monitor but not to a television set. Rather than video, a television accepts RF (Radio Frequency) modulated by video. This means that you can use the Apple IIe with a television set, but the input to the television must be an RF signal modulated by Apple IIe video. Generation of the RF signal and modulation is accomplished in a user supplied modulator. Another name for the user supplied modulator is a pain in the neck.

There are three basic Apple IIe video display modes: **TEXT**, **LORES** graphics (LOw RESolution colored blocks) and **HIRES** graphics (HIgh RESolution colored points). Additionally, LORES and HIRES graphics can be displayed with four lines of text at the bottom of the screen in the Apple IIe MIXED mode. MIXED mode is very useful, as far as it goes, because there are many times when the graphics programmer needs to enhance a display with text. However, four lines at the bottom turn out to be inadequate for many purposes. The HIRES screen has good enough resolution to draw text, and several programs are available that make it relatively easy to place upper/lower case text on the HIRES screen. This type of text can be drawn alongside graphics to enhance graphic displays.

All display modes can be switched to normal horizontal resolution (40 TEXT characters, 40 LORES blocks, or 280 HIRES points) or double horizontal resolution (80 TEXT characters, 80 LORES blocks, or 560 HIRES points). The **SINGLE-RES** (single horizontal resolution) modes are identical to the display modes of the older Apple II computer. The **DOUBLE-RES** (double horizontal resolution) modes offer twice the characters, blocks, or points per horizontal display width as do the SINGLE-RES modes.

Memory scanning is used to generate video in all Apple IIe display modes. Data that represents the display is stored (mapped) in RAM so that video is generated by processing data that comes from RAM as it is scanned repeatedly. Certain areas of RAM are designated as display memory. The designated areas are:

TEXT/LORES	Page 1	$400—$7FF	(1K RAM)
TEXT/LORES	Page 2	$800—$BFF	(1K RAM)
HIRES	Page 1	$2000—$3FFF	(8K RAM)
HIRES	Page 2	$4000—$5FFF	(8K RAM)

As an example, assume that the computer is in TEXT mode, page 1. Then memory in the range $400—$7FF will be scanned approximately 60 times a second and the data in that memory area will be processed for video output. Part of display memory is always being scanned while the computer is on. The Apple IIe is designed so that this constant scanning satisfies the refresh requirement of the dynamic RAM.

Page 1 and page 2 are primary and secondary memory display areas that are switched via the PAGE2 IOU soft switch. Page 1 is normally selected in all modes (PAGE2 soft switch reset), but use of page 2 may suit the programmer's purpose.

An important consequence of the Apple IIe display implementation is that the video display steals memory from the user. The programmer must program around the display areas if he intends to use the associated displays.

SINGLE-RES displays are mapped in motherboard RAM only. One byte of the display map is processed for each cycle of the MPU, and 40 bytes of the display map are scanned to process the displayed portion of a single horizontal scan of the television or monitor. Based on the number of bytes that make up the displayed portion of a horizontal scan, the SINGLE-RES TEXT, LORES, and HIRES modes will be referred to in this book as the **TEXT40**, **LORES40**, and **HIRES40** modes when it is necessary to distinguish them from their DOUBLE-RES counterparts. For the reason made clear in the next paragraph, the DOUBLE-RES TEXT, LORES, and HIRES modes will be referred to as the **TEXT80**, **LORES80**, and **HIRES80** modes.

DOUBLE-RES displays are mapped in motherboard RAM and auxiliary card RAM. For every MPU cycle, first one byte of the auxiliary card display map, then one byte of the motherboard portion of the display map are processed to generate video.

A total of 80 bytes of the overall display map are scanned to process the displayed portion of a single horizontal scan of the television or monitor. If these are numbered 0—79, the even bytes are stored in auxiliary card RAM, and the odd bytes are stored in motherboard RAM.

A RAM card must be installed in the auxiliary slot to utilize the DOUBLE-RES modes of the Apple IIe. A 64K auxiliary RAM card enables use of all DOUBLE-RES modes, and a 1K auxiliary RAM card enables use of only the TEXT80 mode (80-column text).* Additionally, a 1K auxiliary RAM card enables use of LORES80 mode if a wire is connected between pins 50 and 55 of the 1K RAM card's edge connector.

Scanning for video output is not performed by the MPU but by the IOU. Inside the IOU, there is a counter whose outputs are used to make up the video RAM address, a television sync signal, and other video related signals. This counter synchronizes the television scan to its addressing of RAM and can be thought of as scanning RAM while it scans the television (or video monitor) picture. Consequently, it is referred to in this book as the **video scanner**.

The scanner accesses RAM in a way that is completely transparent to the MPU. During the first half of every 6502 cycle period, the video scanner accesses motherboard and auxiliary card RAM. During the second half, the 6502 accesses motherboard RAM, auxiliary slot RAM, or other device. The scanner access to RAM is always a read access and the data which comes from RAM during the scanner access is saved and processed by the **video generator** to make video. The 6502 access can be either read or write and, on some cycles, the 6502 may not access RAM at all.

The programming method for controlling the Apple IIe display is to select the display mode by setting or resetting soft switches, and to compute or look up the memory addresses of screen locations and modify those addresses to achieve the desired display. The video scanner scans the display area determined by the display mode, and the resulting memory data is processed as text or graphics as determined by the display mode.

TEXT characters are represented in the RAM display map as ASCII (American Standard Code for Information Interchange). In addition to ASCII, code for normal display (white on black), inverse display (black on white), or flashing display (alternating normal and inverse) are stored for each text character. One character is stored per byte of display memory. As text is scanned, the coded data from memory is translated to 5 x 7 dot matrix video in normal, flashing, or inverse format. There are 96 displayable upper case, lower case, numeric, punctuation, and special text characters, all of which can be displayed in normal or inverse format and 64 of which can be flashed between normal and inverse format. The TEXT display is 40 columns by 24 lines in SINGLE-RES mode and 80 columns by 24 lines in DOUBLE-RES mode.

The 80-column text capability of the Apple IIe is implemented in hardware and in firmware so that the Apple IIe emulates an Apple II with an 80-column card installed in Slot 3. This emulation is carried out to such an extent that the Apple IIe is, in fact, a 40-column display computer with a peripheral 80-column capability. The Apple IIe powers in 40-column mode, and it will remain in that mode up until a program, maybe or maybe not guided by operator input from the keyboard, selects the 80-column mode.

LORES graphics is a programmable display of 40 columns and 48 rows (SINGLE-RES) or 80 columns and 48 rows (DOUBLE-RES) of colored blocks. Each block can be any one of 15 colors including black and white. Apple claims 16 colors but the two grays are identical in color and luminance. There are, however, 16 different LORES patterns, even though they produce only 15 discernible colors, and these will be referred to as the 16 LORES colors.

LORES is mapped in the same display area as TEXT, so memory scanning is identical in the two modes. In LORES, rather than converting ASCII to video, the video generator processes the bit pattern directly into video. The code for each LORES block requires four bits, so there is code for two blocks in every byte of display memory. Also, there is a direct correspondence between the screen location of a pair of LORES blocks and one text character as shown in Figure 1.2.

HIRES40 graphics mode is a programmable array of 280 columns and 192 rows of dots. Because of the way video is generated in the Apple IIe, the color of any dot is dependent on its horizontal position. To draw a violet horizontal line, for instance, every other dot in one row is turned on. To draw a violet figure, only half of the columns of dots can be turned on. This is also true of the other HIRES40 colors: green, orange, and blue. There is only 140 x 192 resolution when drawing these four colors.

*The DOUBLE-RES graphics modes are not available on Revision A motherboards.

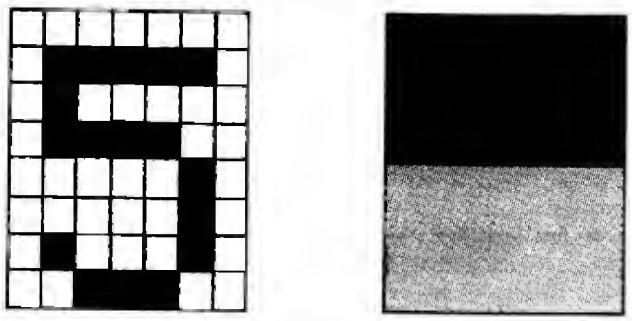


Figure 1.2 TEXT and LORES Graphics.

White (the absence of color) and black (the absence of luminance) can also be displayed. The 280 dots in any row are divided into 40 groups of seven dots. Each group of seven dots may be shifted together horizontally one half of a dot position, changing the colors of any colored dots in that group of seven. Thus, there are 560 horizontal dot positions in each row, but only 280 dots are independently programmable.

HIRES80 graphics mode is a programmable array of 560 columns and 192 rows of dots. Each dot in the array is independently programmable, and the horizontal resolution is so fine that all 16 LORES colors can be produced, and no shifting of 7-dot patterns is required or available. Resolution is 560 x 192 in monochrome plotting, and varies from 140 x 192 to 560 x 192 in 16-color plotting depending on color.

This brief statement of HIRES graphics capabilities is probably just enough information to let the reader know that the subject of HIRES is complex. Full understanding is possible in the light of more detailed analysis, and HIRES is covered in greater detail in Chapter 8. For now, let the resolution of the Apple IIe HIRES display be summarized as varying from 140 x 192 to 560 x 192 depending on color or monochrome plotting and selection of HIRES40 or HIRES80 mode.

The HIRES memory display area is much larger than the TEXT/LORES area: 8192 bytes of motherboard RAM for a HIRES40 display, and 8192 bytes of motherboard RAM and 8192 bytes of auxiliary card RAM for a HIRES80 display. This is the hardware cost of high resolution.

The Keyboard

The keyboard is the primary human input to the Apple IIe (as opposed to storage media input such as cassette or disk). Virtually all human alphanumeric input is via the keyboard, and the MPU of the Apple IIe spends the majority of its life cycling through a little firmware routine called KEYIN (or GETKEY if the 80-column firmware is active). This routine samples the keyboard to see if a key has been pressed, while incrementing a random number counter and occasionally flashing the screen cursor. KEYIN checks the keyboard at a rate of about a 165 million times an hour, and if anyone asks you what an Apple does, you can answer "mainly, it checks to see if a key has been pressed."

Enough silliness. The keyboard has 63 keys that represent letters of the alphabet, numbers 0—9, punctuation characters, symbolic characters, and special functions. These keys are arranged like those of the keyboard of an IBM *Selectric* typewriter. An auto repeat function (mechanized in the IOU) simulates rapid keypresses when a key is held down constantly, and provisions exist for programs to determine when a key is being pressed or when a key has been pressed. Apple IIe keys "roll over", meaning that if one key is held and another is pressed, the newly pressed key will be read by the controlling program.

Most of the keys produce ASCII which can be read by a program, and most of the ASCII keys, including the alphabetic keys, produce shifted ASCII if the left or right SHIFT key is held down simultaneously with the ASCII producing key. Since the keyboard input and text output are both ASCII, it is fairly easy to output characters to the video display as they are entered from the keyboard. This is done by keyboard input and video output routines in the Apple IIe firmware.

Special function keys on the keyboard are ESC, DELETE, RESET, TAB, CONTROL, RETURN, SHIFT, CAPS LOCK, open Apple, close Apple, left

arrow, right arrow, down arrow, and up arrow. CONTROL and SHIFT modify the ASCII produced by other simultaneously pressed keys while CAPS LOCK is a 2-position locking switch that forces upper case ASCII from the alphabetic keys when it is latched in the down position. RESET is tied to Apple IIe RESET' line, and if CONTROL and RESET are pressed simultaneously, RESET' drops low to reset the Apple IIe. Resetting the Apple IIe consists of resetting the 6502, all MMU soft switches, most IOU soft switches, and all peripheral cards that respond to RESET'.

ESC, DELETE, TAB, RETURN, left arrow, right arrow, down arrow, and up arrow produce ASCII which must be interpreted by the controlling program. The codes for ESC and DELETE are unique, but TAB, RETURN, left arrow, right arrow, down arrow, and up arrow produce code that is identical to that of CONTROL-I, CONTROL-M, CONTROL-H, CONTROL-U, CONTROL-J, and CONTROL-K respectively.

The open Apple and close Apple keys are not associated with other keyboard functions. Instead, these are connected to the PB0 and PB1 serial inputs described in the next section. Pushing open Apple or close Apple is equivalent to pushing pushbutton 0 or pushbutton 1 on a paddle set or joystick, and these keys are mounted on the keyboard only to provide a convenient means of activating the PB0 and PB1 input lines.

All ASCII produced by Apple IIe keypresses comes from the **keyboard ROM** which is a standard 2K ROM. This ROM contains ASCII for a standard keyboard layout and an alternate keyboard layout. The alternate layout is a Dvorak layout* in American Apple IIe's. In export versions, it is usually a layout tailored to the requirements of the host country's primary language. The alternate layout can be selected by installing a switch assembly as shown in an application note at the end of Chapter 7.

While there is no numeric keypad built into the Apple IIe keyboard, there is a jack on the motherboard which accepts a numeric keypad. Like ASCII from the main keyboard, ASCII from this external keypad comes from the keyboard ROM, so the keys of a keypad can be defined as desired by installing a customized keyboard EPROM on the motherboard.

Other I/O

I/O is Input/Output. Our point of reference for this discussion is the motherboard, meaning that we speak of input to the motherboard and output from the motherboard. The peripheral slots give the Apple IIe an extremely versatile I/O capability, but there is a good deal of additional I/O circuitry built into the Apple IIe. The keyboard input and video output are the most significant motherboard I/O. There are also some useful serial I/O ports.

Serial data is data on one line. This is opposed to **parallel** data on more than one line (eight lines, for instance). To transfer eight bits serially, each bit of information is placed on the same line one after another. This takes eight times as long as an 8-bit parallel transfer, but requires only one connecting wire. The keyboard is a parallel input. The video is not a simple digital output but a mildly complex signal output with a serial data component. In addition to these I/O capabilities, there are **eleven serial I/O ports** and **four resistance sensitive timer inputs.**

The **speaker** output is a serial output port connected to a speaker through an audio amplifier. The **cassette** input and output are serial data transmitted via audio phone jacks on the motherboard accessible from the back of the case. They are designed to connect directly to the earphone output and microphone input of a common audio tape recorder. Firmware routines in motherboard ROM read and write cassette data in Apple's storage format.

Usage of 5 ¼ inch floppy disks is so prevalent that cassette storage is rarely used by most Apple owners. Floppy disk I/O is not a built-in capability of the motherboard, so the disk electronics are contained in the drive and on a peripheral card called the disk **controller**. Disk data is transferred in parallel between the MPU and the controller, and serially between the controller and the drive. Control of disk I/O requires an extensive program, and the most commonly used program of this nature is **DOS 3.3** (Disk Operating System, version 3.3), a product of Apple Computer, Inc. A more recently developed DOS, and the one which is the current focus of support by Apple, is called **ProDOS**.

The other serial I/O signals are **TTL** (Transistor Transistor Logic) compatible. TTL is a very common logic family of integrated circuits used for digital logic. The logic devices on the Apple IIe motherboard are either TTL or interface directly to TTL.* TTL devices operate with two voltages corresponding to the two states of digital logic. The

*Dvorak is keyboard layout designed to permit faster typing than is possible in the conventional QWERTY layout.

*Most TTL chips in the Apple IIe are LSTTL (Low Powered, Schottky-Barrier diode clamped TTL). The 6502, ROM, RAM, the MMU, the IOU, and the keyboard decoder are TTL compatible MOS (Metal Oxide Semiconductor) chips.

TTL low voltage is 0 to 0.8 volts, and the TTL high voltage is 2.4 to 5 volts. These are the two voltage levels which represent digital information throughout the Apple IIe.

There is a 16-pin DIP (Dual In line Package) socket on the Apple IIe motherboard which is generally called the **game I/O connector**. A set of two paddles, a joystick, or a resistive graphics pad is normally connected here, but there is a capability for multiple uses. Four of the pins are **annunciator** outputs. These are output lines which can be independently switched to a TTL high or low level by the controlling program. A fifth TTL output is called a **strobe**. This output is high unless a program triggers it. It then goes low for just 0.5 microseconds (half of a 6502 cycle), then returns to its normal high state.

There are three TTL input ports on the game I/O connector which can be read by a program. Two of these, PB0 and PB1, are normally connected to **pushbuttons** on the joystick, paddles, or graphics pad. PB0 and PB1 are also connected to the keyboard open and close Apple keys respectively. Additionally, if the motherboard X6 jumper is soldered, the SHIFT' line is connected to PB2 so that the left and right SHIFT keys activate this third game I/O TTL input.

The **paddles** themselves are just potentiometers (variable resistors). **Joysticks** are two potentiometers mechanically linked so that the resistance of one potentiometer represents horizontal motion and the resistance of the other potentiometer represents vertical motion. Game I/O **graphics pads** consist of X-ordinate and Y-ordinate resistive surfaces arranged and wired so that the X and Y resistances vary with the point on the pad at which pressure is applied.

In addition to the game I/O socket, the four timer (paddle) and three TTL (pushbutton) inputs are connected to a **game I/O extension jack** in the back of the Apple IIe. This 9-pin jack provides a means of connecting a paddle set, joystick, or other device to the Apple IIe without lifting the cover. Furthermore, when a device is connected to this extension jack, the game I/O lines which are not used by the extension jack device are available at the game I/O socket for connection to other devices.

The Power Supply

Household power measures from 100 to 220 Volts AC (Alternating Current), depending on the country in which the house is located. Most of the circuits in the Apple IIe, however, require +5 volts DC (Direct Current) referenced to ground (0 volts). Converting relatively high voltage, household AC power to the required low voltage DC power required by the Apple IIe is the function of the **power supply**.

The power supply in an Apple IIe is designed to operate on the household power in the country in which it is sold. In any country, the Apple IIe power supply generates +5, −5, +12, and −12 volts DC referenced to ground. These voltages are distributed throughout the motherboard to any device that needs them. Additionally, all four voltages and ground are available at the peripheral slots to supply power to peripheral cards, and +5 VDC and ground are available at the auxiliary slot to supply power to an auxiliary card.

SUMMARY

The Apple IIe is a single board, 6502 based microcomputer with built-in memory and video generation circuitry. It is an improved version of the older Apple II computer. Enhancements include full upper and lower case text handling capability, 80-column text video display, and 128K of motherboard and auxiliary slot RAM, as opposed to the upper case only, 40-column, 48K Apple II.

The Apple IIe circuit board contains seven peripheral slots and an auxiliary slot which hold smaller boards, and it is therefore thought of as a motherboard. The slots give the Apple IIe expansion and I/O capabilities comparable to more expensive card cage microcomputer designs.

The motherboard can be one of two versions—one which outputs video that is color compatible with the NTSC television system used in America, or one which outputs video that is color compatible with the PAL television system used throughout western Europe except in France. An Apple IIe in a given country will contain the version of the motherboard compatible with that country's television system. Additionally, the video and keyboard RAM will be tailored to the requirements of that country's language or languages.

The 6502 in the Apple IIe operates at 1.0205 MHz. IRQ', NMI', RESET', and READY signals to the 6502 are connected to the peripheral slots. The DMA' signal enables peripheral cards to isolate the MPU from the rest of the motherboard. This enables control of the Apple IIe from secondary MPUs or other DMA devices in the peripheral slots. MPU control of the various hardware features is via address decoding.

The motherboard contains 65,536 bytes of dynamic RAM, and motherboard circuitry fully supports an additional 65,536 bytes of dynamic RAM in an auxiliary slot RAM card. 16,128 bytes of firmware include Applesoft BASIC and a system monitor containing a number of important utilities.

In addition to the MPU, RAM, and ROM, there are three important special purpose ICs that implement Apple IIe motherboard logic functions. The MMU controls the overall configuration of the Apple IIe memory map; the IOU performs multiple functions related to generation of the video display and other I/O; and the timing HAL contains most of the circuitry required for the generation of Apple IIe timing signals. The controlling MPU program manipulates overall memory configuration, the video display mode, and some I/O functions by setting or resetting programmable MMU and IOU soft switches.

The video output is compatible with a video monochrome or color monitor. It can be used with a home TV when connected through an inexpensive modulator. Either single or double horizontal resolution displays can be produced by programs, although an auxiliary slot RAM card is necessary for use of the DOUBLE-RES display modes. All DOUBLE-RES displays are available with a 64K auxiliary RAM card, but a 1K RAM card only supports DOUBLE-RES text or, with a minor RAM card modification, DOUBLE-RES LORES graphics.

TEXT is upper and lower case, 5 x 7 dot matrix representation in a 40 character by 24 line (SINGLE-RES), or 80 character by 24 line (DOUBLE-RES) display. There are 96 video text characters, all of which can be displayed normally (white on black) or inverted (black on white). Sixty-four of the text characters can be flashed between normal and inverse display. This includes numerals, punctuation, and upper case alphabetic characters but excludes lower case alphabetic characters.

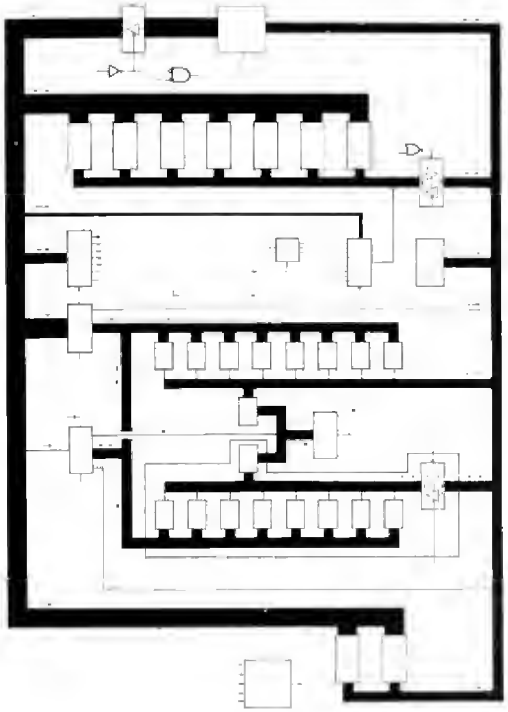
Graphics modes include 40 x 48 (SINGLE-RES) and 80 x 48 (DOUBLE-RES) LORES block modes in 15 colors, 140 x 192 HIRES point mode in six colors (SINGLE-RES), 280 x 192 HIRES point mode in black and white (SINGLE-RES), 140 x 192 HIRES point mode in 15 colors (DOUBLE-RES), 140 x 192 to 560 x 192 HIRES point mode in 15 colors with color dependent resolution (DOUBLE-RES), and 560 x 192 HIRES point mode in black and white (DOUBLE-RES). Some capabilities exist for mixing text and graphics.

The video display in all modes is mapped in certain areas of RAM, motherboard RAM in the SINGLE-RES modes, and both motherboard and auxiliary card RAM in the DOUBLE-RES modes. IOU circuitry continuously scans one of four possible areas in motherboard and auxiliary card RAM while RAM output is processed to generate video. RAM addressing is time shared between the system address bus and the IOU video scanner. 6502 access to RAM alternates with video scanner access so, while the 6502 operates at 1 MHz, motherboard and auxiliary card RAM are accessed at 2 MHz. In the process of scanning RAM for video output, the RAM is refreshed.

In addition to video output and the I/O capabilities inherent with the peripheral slots, there are a cassette input port, a cassette output port, a speaker, four TTL control outputs, one .5 microsecond TTL output strobe, four resistance sensitive timer inputs, three TTL inputs, a keyboard, and a numeric keypad jack. Two of the TTL inputs can be activated by pressing the open or close Apple switches on the keyboard.

The keyboard contains 63 key switches arranged like those on an IBM *Selectric* typewriter, and is adequate for most text processing functions. Operational features include a CAPS LOCK key, n-key rollover, and automatic simulation of rapid keypresses when a key is held down (auto repeat). An alternate keyboard layout is electrically selectable, but a switch assembly must be installed to access the alternate layout. Also, because of the versatile nature of the motherboard keyboard circuitry, the keyboard layout can be changed by simply replacing a ROM on the motherboard.

The built-in Apple IIe power supply provides +12, −12, +5, and −5 volts DC referenced to ground. These voltages and ground (0 volts) are distributed throughout the motherboard and to the seven peripheral card slots. +5 volts and ground are also connected to the auxiliary slot.



chapter 2

The Bus Structure of the Apple IIe

There are many signals distributed throughout the Apple IIe, but the most fundamental data transfer takes place on the **data bus**, and the most basic control information is distributed via the **address bus**. To understand how the Apple IIe and other microcomputers really work, it is very important to understand the bus structure. Fortunately, it's not that hard to understand. The basic concepts of the bus structure are within the grasp of nearly everyone who uses a microcomputer.

The bus structure is a natural starting point for learning what really goes on inside the Apple computer. Discussing the bus structure will lead naturally to the discussion of the other microcomputer elements that the bus is connected to. First, though, we need to find out what a bus is and how it is used.

COMPUTER BUSES AND THREE STATE LOGIC

Logic signals in the Apple are distributed electrically via conductive paths on the motherboard. When a number of signals are grouped functionally and distributed throughout a microcomputer, they are collectively referred to as a bus. Physically, then, a bus is an electrical distribution of multiline information. In the Apple, the **address bus** is a **sixteen**-line electrically distributed information group, and the **data bus** is an **eight**-line electrically distributed information group.

Some devices connected to a bus are strictly receivers of information. ROM is like this in its connection to the **address bus**. Receivers respond to the high/low information on the lines of the bus without appreciably affecting the bus information. Electrically speaking, the receiver input presents a high impedance to the bus which enables other devices to bring the bus lines high or low. If impedance is a new word to you, it may help to think of high impedance as high isolation.

Some devices on a bus must be information transmitters capable of bringing the bus lines high or low. If more than one information transmitter is connected to a bus, each transmitter must be able to disconnect itself from control of the bus by presenting a high impedance to the bus. Only one device can control the bus at a time. Instead of two state, the outputs of these devices are said to be three state or

tri-state. The three states are high voltage, low voltage, and high impedance. All information transmitters to the data bus of the Apple are capable of presenting these states to their bus connections. The ROM output to the **data bus** is a typical three state output.

A third type of device, capable of transmitting to or receiving from a bus, is called a **transceiver (transmitter/receiver)**. The MPU, for instance, receives (reads) data from and transmits (writes) data to the data bus, so as far as the data bus is concerned, the MPU is a transceiver. While the MPU is reading, it presents a high impedance to the data bus so the addressed device can place data on the data bus. While the MPU is writing, it controls the data bus.

Figure 2.1 shows a hypothetical 4-line bus. The symbols shown are schematic representations of a tri-state line driver, a line receiver, and a line transceiver. A triangle represents a single line driver. Triangles with a control line coming in from the side are tri-state line drivers. A little circle at a control input to a triangle means that the input is active when its voltage is low. Here is a truth table for the tri-state line driver shown in Figure 2.1:

INPUT	OUTPUT ENABLE	OUTPUT
Any	Low	High Impedance
High	High	High
Low	High	Low

The control line either enables the high/low output or forces the output to high impedance. The high/low output, when enabled, follows the input.

It can be seen that the **output enable** controls of the various information transmitters are the key to cohesive control of the bus. For a bus with many possible information transmitters, like the data bus of the Apple, there has to be some intelligent management of the various tri-state output enables. We will see shortly how this is accomplished. In the following discussions, remember that when a device like a ROM chip responds to an address prompt by placing data on the data bus, this is accomplished via an output enable to the tri-state outputs of the ROM chip.

Figure 2.2 shows a highly simplified diagram of the bus structure of the Apple IIe. There are two distinct multiline signal paths: the **address bus** and the **data bus**. The R/W' line (Read/Write control) is shown separate and can be thought of as an extension of the address bus controlling the direction of data flow on the data bus. Communication takes place on every 6502 cycle between the MPU and an addressed device. Data flows between the MPU and the device in a direction determined by the R/W' line. The MPU controls the R/W' line and the address bus.

Figure 2.3 shows the two types of bus access which occur in the Apple IIe. In a read access, the MPU places an address on the address bus and reads the data bus. In a write access, the MPU places an address on the address bus and places data on the data bus. This establishes a system of data bus control that had to be implemented in the design of the Apple. The control system works like this:

1. When the R/W' line is low (write access), all inputs to the data bus are disabled except the MPU.
2. When the R/W' line is high (read access), all inputs to the data bus are disabled except the device which is addressed.

This system coneept keeps traffic flow orderly and is a basic feature of microcomputer design.

The only remaining points to be made about buses involve semantics. The **peripheral slots** are sometimes referred to as the **peripheral bus** or the **Apple bus**. In fact, the wiring of the slots fits our description of a bus as a functional group of distributed signals. The slots are a bus whose distributed signals include the address bus, the data bus, and other signals. Up to this point, the discussions have avoided calling the slots a bus only to avoid confusion between the card cage bus and the more basic address bus and data bus. The connections to the RAM and ROM chips form two more distributed signal groups that can be referred to accurately as the RAM bus and the ROM bus. This book will continue to use the word "bus" to refer to the address bus, the data bus, and the extensions of these two basic communications paths. The peripheral bus, RAM bus, ROM bus, and other distributed signals will be referred to using other terminology.

The lines of the various buses in the Apple are referred to by one or more letters followed by a number. For example, the lines of the Apple address bus are referred to as A0 through A15. The largest number, A15 in this example, refers to the line which carries the most significant bit of information. A list of bus terminology used in this book follows here.

LINE DRIVER
Information is transmitted to the bus by a device with tri-state outputs.

LINE RECEIVER
An information receiver presents a high impedance to the bus.

LINE TRANSCEIVER
A bidirectional connection to the bus must present a high impedance to the bus when in receive mode.

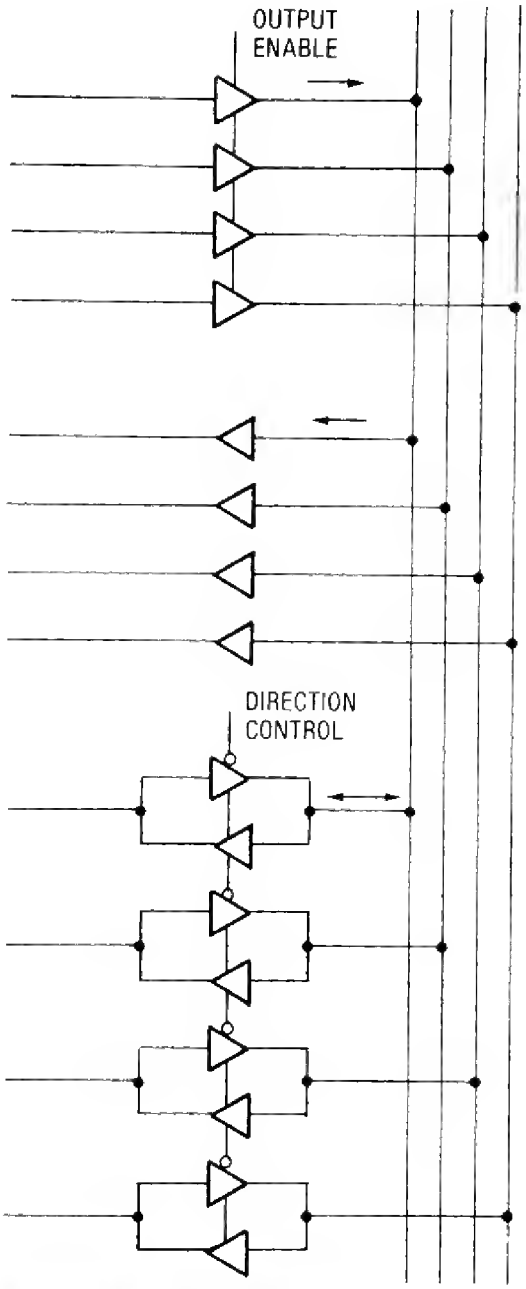


Figure 2.1 A Hypothetical Four-Line Bus.

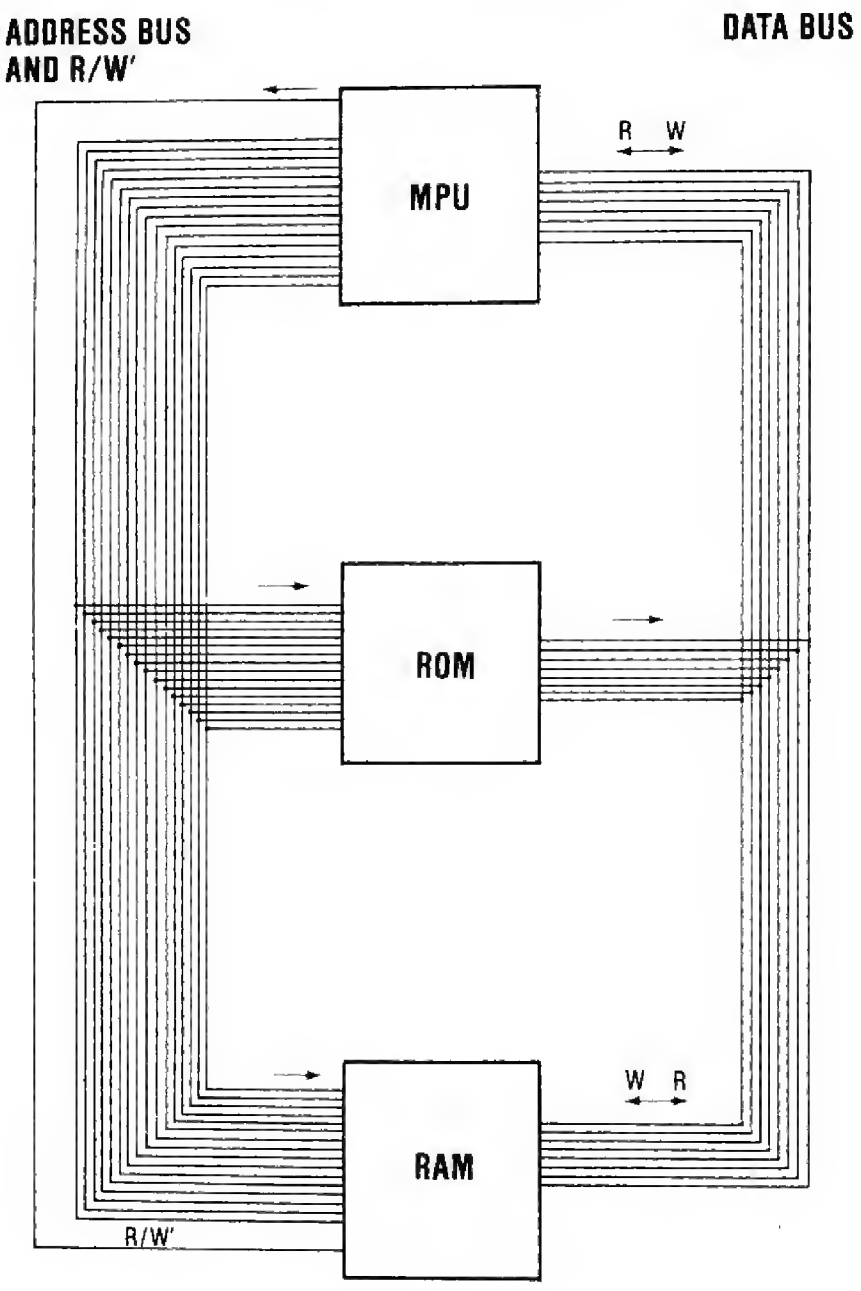


Figure 2.2 Basic Microcomputer Building Blocks.

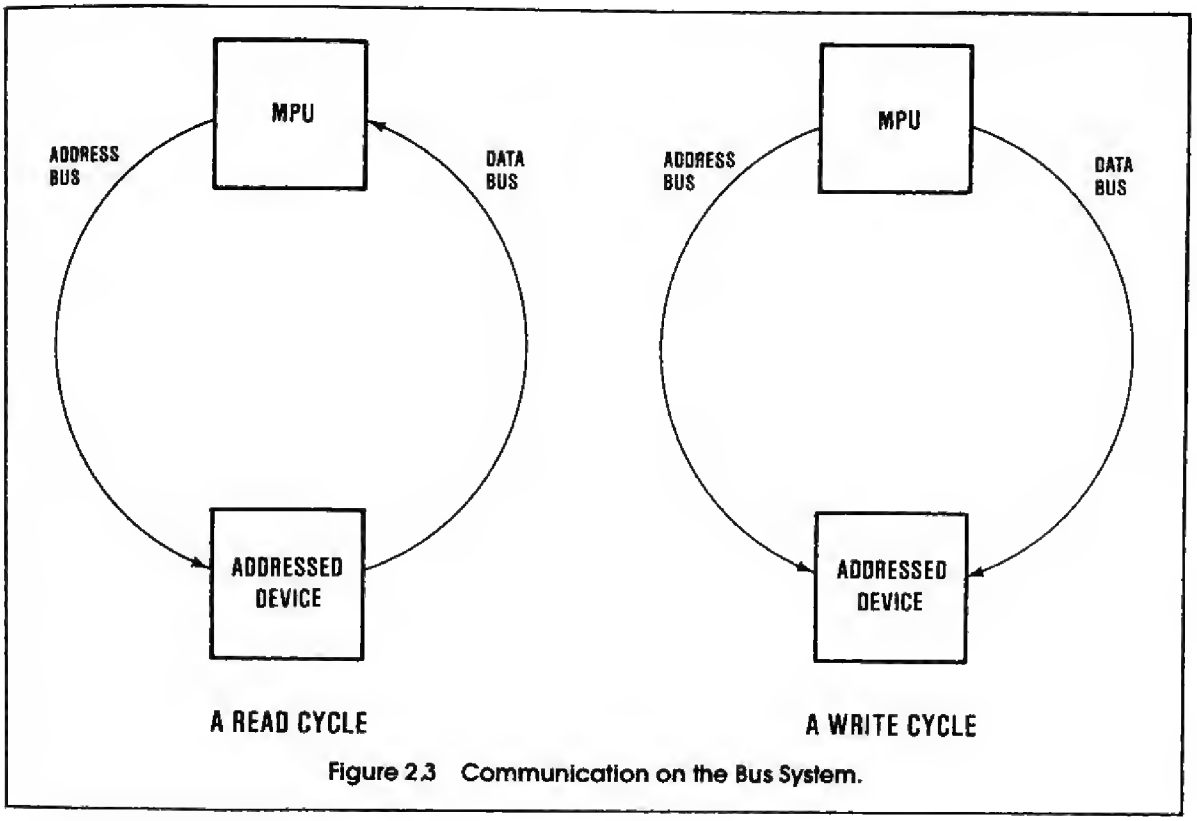


Figure 2.3 Communication on the Bus System.

NAME OF BUS	LINE TERMINOLOGY
Address	A0—A15
Data	MD0—MD7*
Multiplexed RAM address	RA0—RA7
Auxiliary RAM data	AUXD0—AUXD7
Video data	VID0—VID7
Peripheral slot data	D0—D7

By this time the reader should understand the concept of the bus as a communication path. We will now move on to how microcomputers in general and Apples in particular perform their functions in a bus environment.

THE PIGEONHOLE COMPUTER

There is an old analogy for understanding digital computer operation which you don't see often enough in personal computer instruction literature. It possibly is not that helpful for understanding BASIC programming, but it is very much like the way a microcomputer works.

The analogy goes like this. A computer is like a gigantic row of pigeonholes with pieces of paper in them. Each piece of paper has an instruction on it. There is a man who goes to each pigeonhole, one after the other, reading the instructions and doing what they say. The man always gets the next instruction from the next pigeonhole in the row unless an instruction tells him to go to some other pigeonhole.

That's the pigeonhole computer. The man is executing a stored sequential program. The man is the microprocessor. The row of pigeonholes is computer memory. The instructions are the program. The microprocessor is smart enough to sequence through memory and do what it's told, but it has to be told. It has to have a program.

*MD in MD0—MD7 stands for the MOS Data bus. Apple chose this nomenclature because most of the ICs connected to MD0—MD7 are MOS ICs.

THE MPU, RAM, AND ROM

The microprocessor is the engineering marvel which made all the home computers possible. The 6502 MPU is what executes the programs in the Apple. Viewed from the outside, its capabilities include manipulation of the address bus and R/W' (Read/Write') control, writing data to the data bus, reading data from the data bus, logical and arithmetic manipulation of data, and response to various control inputs. All of these add up to execution of a sequential program that comes from the data bus.

You see, the man from the pigeonhole computer resides inside the MPU. The little guy has this control line called R/W', and he can put any address from 0 to 65535 on the address bus. He uses the R/W' line to tell the outside world whether he's reading from or writing to the data bus. He uses the address bus to tell the world where he wants the read data to come from and the write data to go to. There are plenty of things this man can do, but his most favorite thing in the whole world is to increment the address bus and read the results on the data bus. While he's reading, this little workaholic interprets the data he reads as instructions. If there is an outside device that is responding to his address prompts with a valid sequential program, he will flat out execute the program. This means that you can exploit his insatiable reading appetite and get him to do what you want if you're smart enough. That's all any microcomputer designer ever really expects from an MPU.

The key requirement above was an outside device responding to the address prompts. This device is memory: ROM or RAM. All of the addressing on the Apple address bus is parceled out to various devices. RAM gets addresses $0—$BFFF. ROM and high RAM share $D000—$FFFF, although this range is thought of as being primarily assigned to ROM. The peripheral slots are controlled by $C090—$CFFF. $C000—$C08F is divided up among the keyboard and cassette and all the other built-in devices. If the 6502 happens to be executing a program in the $D000—$FFFF range with high RAM disabled, then **ROM** is responding to the addressing with a series of data which the 6502 is interpreting as a program. If the ROM program tells the MPU to store a byte of data at $400, the MPU takes a microsecond to bring R/W' low, set the address bus to $400, and place the pertinent data on the data bus. The data is accepted by address location $400 which is in **RAM**. That pigeonhole of RAM owns address $400 just as sure as your mailbox has a unique mailing address. Inside RAM, inside ROM, all along the address bus, address decoding takes place every 6502 cycle to enable only one of 65536 possible addresses.

The 6502 is continually executing a program while power is applied. If it gets lost and tries to execute a program where no program exists, it interprets whatever jibberish is appearing on the data bus as a program and executes it anyway. An unstoppable program-executing machine like this has to have a starting point when you turn the computer on. It also needs a way to start from scratch when it gets lost. This starting point is the RESET' input to the 6502.

The RESET' input to the 6502 goes low when the RESET key is pressed, when a peripheral card makes it go low, or when the computer is turned on. Any one of these occurrences makes the 6502 stop what it's doing, load the address of the next program step from locations $FFFC and $FFFD, and start executing at that address. The contents of $FFFC and $FFFD are the low and high bytes of the reset vector.*

The $FFFC/$FFFD reset vector comes from motherboard ROM since the high RAM is disabled for reading by the reset sequence. In Apple IIe ROM, the contents of $FFFC/$FFFD is $FA62, the address of the firmware reset routine. There are several important aspects of this routine that determine features of the Apple IIe, but the important point here is that the Apple has a power-up routine in ROM. This is an essential feature of microcomputer design. You might say it guarantees that the 6502 always gets out of bed on the right side.

Another routine which a microcomputer always has in ROM is a routine to load data from a storage device into RAM so that execution of saved programs is possible. The Apple IIe, however, has much more than the bare necessities in its 16K of ROM space. The naked Apple is a cassette based system in which BASIC in ROM and a system monitor in ROM prevent unnecessary user aging while waiting for the computer to become operational at turn on. Additionally, firmware diagnostic routines are available to confirm correct operation or aid in fault isolation in case of hardware failure.

*Two 8-bit RAM locations are required to store a 16-bit 6502 address. The 6502 fetches a 16-bit address from an adjacent pair of memory locations. The less significant byte of the address is fetched from the lower memory location, and the more significant byte is fetched from the higher memory location.

RAM ADDRESSING AND DATA DISTRIBUTION

While Figure 2.2 adequately depicts the fundamental MPU access to RAM, it does not show many details of the layout of RAM in the Apple. In fact, there is some complexity to RAM access. This is due to the nature of the dynamic RAM chips, the dual access to RAM from the MPU and the video scanner, and the multifaceted bank switching of RAM that occurs in the Apple IIe. Figure 2.4 is the partial diagram of the Apple IIe's bus structure, expanded from Figure 2.2 to show more of the details of address and data distribution to RAM. The thick black and dark gray lines in Figure 2.4 represent the multiple lines of the address bus and the data bus, respectively. R/W' is considered to be distributed with the address bus. The MPU, as before, is in control of the address bus.

A 64K RAM card is shown installed in the auxiliary slot in Figure 2.4. This reflects the fact that the Apple IIe design supports 128K of RAM and an 80-column text display. Eighty columns of text and 128K of RAM are fully implemented in timing, control, and bus structure. They just forgot to mount the auxiliary RAM on the motherboard.

Figure 2.4 also shows some secondary buses which carry address information or data. These buses are not connected directly to the address bus or data bus, but they can be thought of as extensions of the address bus or data bus. The light gray buses are extensions of the data bus, and the medium gray bus is the **multiplexed RAM address bus**, an extension of the address bus.

The Multiplexed RAM Address Bus

The multiplexed RAM address bus is a solution to a common problem in VLSI (Very Large Scale Integration) IC packaging. The problem is that you can pack such complex and extensive logic functions into a small IC that there are not enough pins on the IC to input and output all the information required to support the logic functions. The solution is to multiplex (switch, share) the information. In the case of Apple RAM, this means multiplexing the information of the sixteen lines of the address bus onto the eight lines of the RAM address bus.

We don't want to get too steeped in RAM addressing right now, but the basic situation is that there are not enough pins on a 64K dynamic RAM chip to address 64K memory cells simultaneously.* The RAM is addressed with a one-two punch. First, half of the address information is input to RAM where it is saved. Then the second half of the address information is input and the data access takes place. Both the first half and the second half of the address are input on the same eight pins of RAM, so sixteen bits of information from the address bus must be multiplexed onto eight lines to effect the one-two punch. This multiplexing is accomplished in the **MMU**, and the multiplexed MPU address is distributed from the MMU to all RAM chips on the motherboard and auxiliary card via the 8-line multiplexed RAM address bus (RA0—RA7).

The two halves of dynamic RAM addressing are referred to as the ROW address and the COLUMN address. This refers to conceptual rows and columns of memory cells inside the RAM chips.

The RAM addressing would be complex enough, but in the Apple, the RAM address lines are doubly multiplexed. Both the MPU and the **video scanner** in the **IOU** must access RAM, so the multiplexed RAM address is connected to the IOU as well as the MMU and the RAM chips. During every 6502 cycle, first the video scanner output, then the address bus contents must be switched on to the multiplexed RAM address bus. Each access is accomplished in two halves (the one-two punch). The RAM address multiplexing is cyclical, resulting in the following repeating pattern of access to the multiplexed address bus:

T1 —Video ROW address (IOU)
T2 —Video COLUMN address (IOU)
T3 —MPU ROW address (MMU)
T4 —MPU COLUMN address (MMU)

The IOU connection to the multiplexed RAM address bus is bidirectional. While the video scanner is addressing RAM, the IOU transmits the video ROW address then the video COLUMN address to the RAM address bus. While the MPU is addressing RAM, the IOU monitors the RAM address bus and receives MPU address information. More specifically, the IOU latches (saves) RA0—RA6 of the MMU ROW address and thus monitors A0—A5 and A7 of the address bus without direct connection to the address bus. By this sleight of hand, the need for seven pins on the IOU is eliminated.

*Throughout this book the word "cell" will be used to refer to a unit of memory that stores one bit of data. The word "location" will be used to refer to eight associated memory cells that hold one byte of data in the Apple IIe.

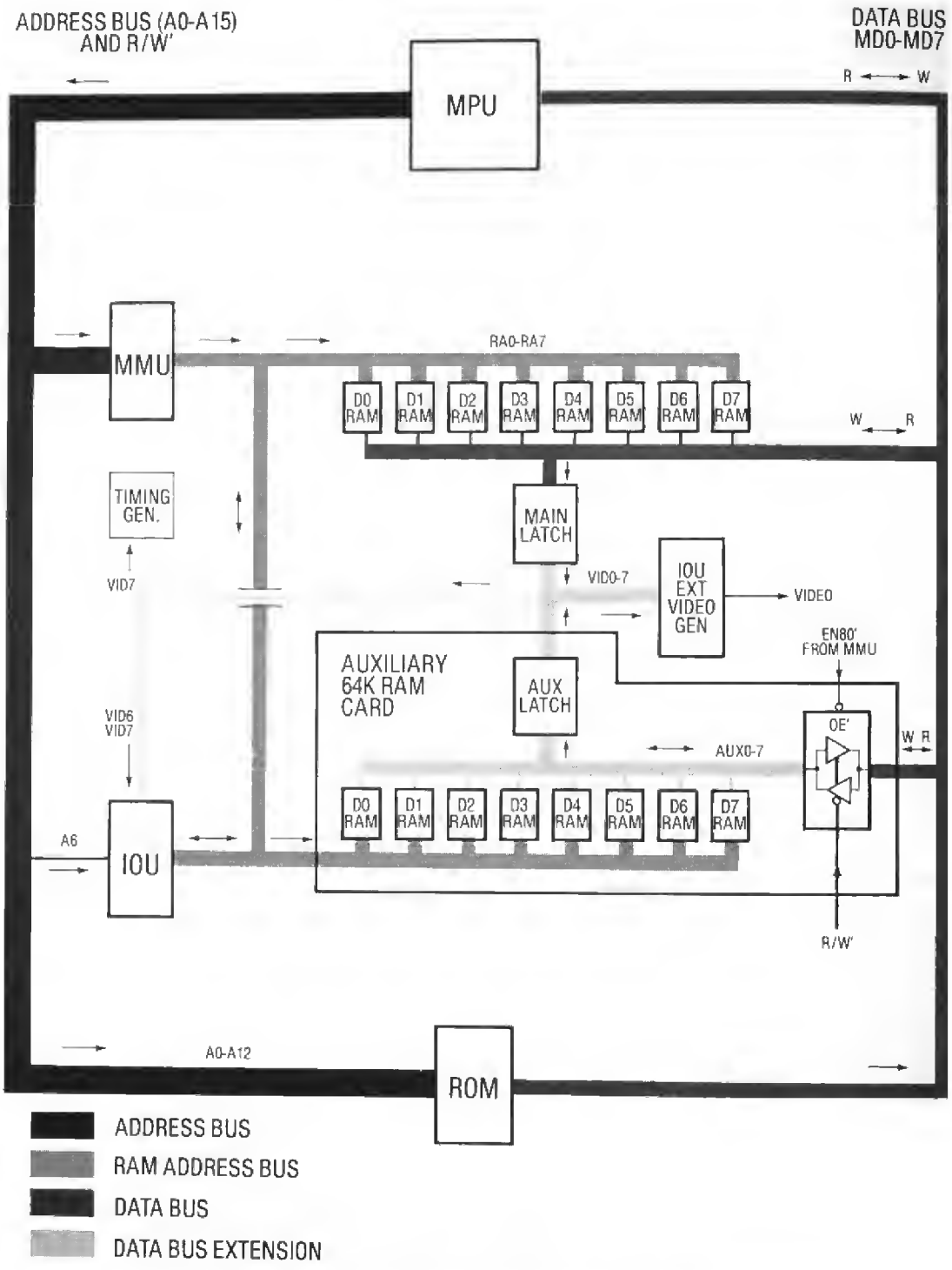


Figure 2.4 Bus Diagram: RAM Addressing and Data Distribution in the Apple IIe.

Video Scanning

The video scanner is not connected to the address bus and is therefore not controllable by the MPU. The scanner is a free running counter **inside the IOU**, completely isolated from program control, that shares RAM on an equal footing with the 6502. The scanner is like a second MPU, but much simpler than an actual MPU. In microcomputer jargon, it is a built-in DMA device performing simultaneous direct memory access with the MPU.

Video scanner access to RAM is a read access as opposed to a write access, but it is of a different nature than MPU read access. The MPU reads data from RAM, meaning that the MPU addresses RAM, and data from RAM comes back to the MPU. In contrast, when the video scanner addresses RAM, the data from RAM does not come back to the scanner. The data goes out, instead, to the video generator for video processing and, in the case of motherboard RAM when R/W' is high, to the peripheral slots via the data bus. As a result, this book does not refer to the video scanner as **reading** data from RAM. Instead, the video scanner is said to **drive** data out of RAM to the video data latches and the peripheral slots.

Other than the fact that the video scanner and MPU both address RAM, their only operational tie is timing. Just as the 6502 executes a machine cycle once every microsecond, the video scanner changes its memory address, and accesses RAM once every microsecond. Logically enough, the timing for the video scanner and MPU originate from the same source. In fact, all timing on the motherboard originates at the same source. The timing involved in the sharing of RAM is quite elaborate and is covered in the chapters on timing generation, RAM, and video generation (Chapters 3, 5, and 8).

The output of the video scanner is used in the IOU for other tasks besides addressing RAM. It is used to make up a number of IOU outputs required in video generation. This includes the sync portion of the VIDEO signal, so the television scan is syncronized with the scanning of RAM. Video scanner outputs are also used in Apple timing generation, MIXED mode switching between GRAPHICS and TEXT, switching between normal and inverse video to create flashing text on the screen, simulating repeated keypresses for the keyboard auto-repeat function, and timing out the power-up reset.

RAM Data Distribution

The 65,536 bytes of motherboard RAM consist of eight 64K dynamic RAM chips. Each RAM chip is organized 64K x 1, meaning that each RAM chip has 65,536 1-bit memory cells, one data input line, and one tri-state data output line. 6502 microprocessor structure requires that memory be organized for 8-bit parallel data transfer, so eight chips provide 65,536 8-cell memory locations in a 6502 system.

Each of the eight motherboard RAM chips is associated with one line of the data bus. The input and output lines of one chip are tied to MD7, the input and ouput lines of another are tied to MD6, etc. The eight RAM chips can thus be thought of as a single 64 kilobyte memory device with eight input/output lines connected directly to the data bus. The R/W' line is gated to the RAM chips when it is time to pass data to or from the MPU, so the RAM chips are able to receive data on MPU write cycles and transfer data on MPU read cycles. The RAM read/write control line is always forced to read when the video scanner is accessing RAM, so the video scanner always reads, never writes.

There is a device connected to the data bus which does not communicate with the MPU. It is the **motherboard video latch.** This latch receives data from the data bus at a point in time when data from the video scanner access to RAM is on the bus. In a sense, then, the data bus is multiplexed. Data travels between the MPU and RAM during MPU access, and data travels from motherboard RAM to the motherboard video latch during video scanner access.

The latched video data is routed, via the video data bus, to **video generation circuitry** both internal (VID6—VID7) and external (VID0—VID5) to the IOU. It is processed there to produce the dot patterns that make up the Apple IIe display. VID7 is also routed to the timing generator where it is used to determine whether or not groups of seven HIRES dots are slightly delayed.

Auxiliary RAM data paths are similar to motherboard data paths with one big difference. The auxiliary RAM data inputs and outputs are not connected directly to the data bus. They are isolated from the data bus by a bidirectional bus driver that only enables data transfer when the MPU is reading from or writing to auxiliary RAM. This creates an **auxiliary RAM data bus** (AUXD0—AUXD7) which is an extension of the motherboard data bus. During video scanner access to auxiliary RAM, the motherboard data bus is isolated from the auxiliary RAM data bus.

There is a latch connected to the auxiliary RAM data bus which saves the data resulting from the video scanner access to auxiliary RAM. Like the motherboard latched video data, the auxiliary

latched video data is routed on the video bus to the video generator for processing to make up the VIDEO signal of the Apple. Both the motherboard latch and the auxiliary latch have tri-state outputs to the video bus, and Apple timing is such that the two latches alternate in controlling the video bus.

The timing involved in scanning RAM for video output is too complex to cover in this chapter. But you should be able to get the general picture from Figure 2.4. In the first half of the MPU cycle, before it is time for the MPU to communicate with the data bus, the video scanner performs a read access to RAM. This access is performed simultaneously in motherboard RAM and auxiliary RAM, and the motherboard data and auxiliary data are driven out together. At a moment when the video data is known to be valid on both the motherboard data bus and the auxiliary RAM data bus, the video data is latched in the motherboard and auxiliary video latches.* For the following half-microsecond, the auxiliary video data is present on the video bus for processing by the video generator. Following that, motherboard video data is present on the video bus for one half-microsecond. At the end of the second half-microsecond, a new set of video data is latched in the pair of data latches. If the Apple is in a DOUBLE-RES display mode, the video generator processes auxiliary and motherboard video data at one half-microsecond per video cycle. If the Apple is in a SINGLE-RES display mode, the video generator ignores the auxiliary data and processes the motherboard data at one microsecond per video cycle.

ADDRESS DECODING

Inside RAM and ROM, some pretty sophisticated address decoding goes on so that data communication is with the correct memory location. Each RAM chip in the Apple IIe has a capacity of 65536 individually accessible bits of information, and each ROM chip has a capacity of 8192 individually accessible bytes of information. Needless to say, much of the circuitry in the memory chips is devoted to decoding the address input.

Like memory, but on a much smaller scale, the Apple must decode addresses to control its various functions. As has been stated previously, the address bus and R/W' line are the way in which the 6502 commands the Apple devices to do things. There are logic circuits in the MMU, the IOU, and some smaller ICs on the motherboard that detect certain addresses or address ranges, then perform control functions or output control signals to various functional areas of the Apple. The following types of control are performed by **address decode:**

1. Gating (enabling) of information to the data bus, including data from serial inputs, peripheral slots, ROM, RAM, the MMU, and the IOU.*
2. Direct control of serial output lines.
3. Control of peripheral slots.
4. Control of display mode soft switches in the IOU.
5. Control of memory management soft switches in the MMU.

Control by address decode gives cohesion to the bus structure.

The address and control functions of the address bus are not separate entities but different ways of looking at the same thing. Addressing memory location $95FF can be thought of as controlling that memory location. Similarly, control of the cassette output line may be thought of as addressing it. The **address bus** could be called the **control bus.**

Figure 2.5 is a partial diagram of the Apple IIe's bus structure highlighting the address decoding motherboard devices. Please refer to this figure during the following discussion.

The primary address decoding circuitry of the motherboard is in the MMU. It alone, of the address decoding elements, monitors all 16 lines of the address bus. The MMU monitors the entire $0000—$FFFF 6502 address range, and activates the other address decoding elements via various control signals. Each ROM chip, for example, is capable of decoding a range of 8192 addresses, but the MMU must tell the ROM chip that it is enabled and an address in its particular range of 8192 addresses is on the address bus. Because it receives an enabling input from the MMU, ROM does not have to monitor all 16 lines of the address bus. It just monitors A0—A12 which is enough to decode a range of 8192 addresses. Similarly, other address decoding elements such as the peripheral decoding circuits and

*As will be seen in Chapters 3, 5, and 8, this moment is PHASE 0 rising. Peripheral cards can also latch the motherboard video data using PHASE 0 rising.

*When a digital signal controls the passage of information in a logic device, it is said to **gate** that information. Gating of information is like opening or closing the gate of a fence to control passage through the gateway.

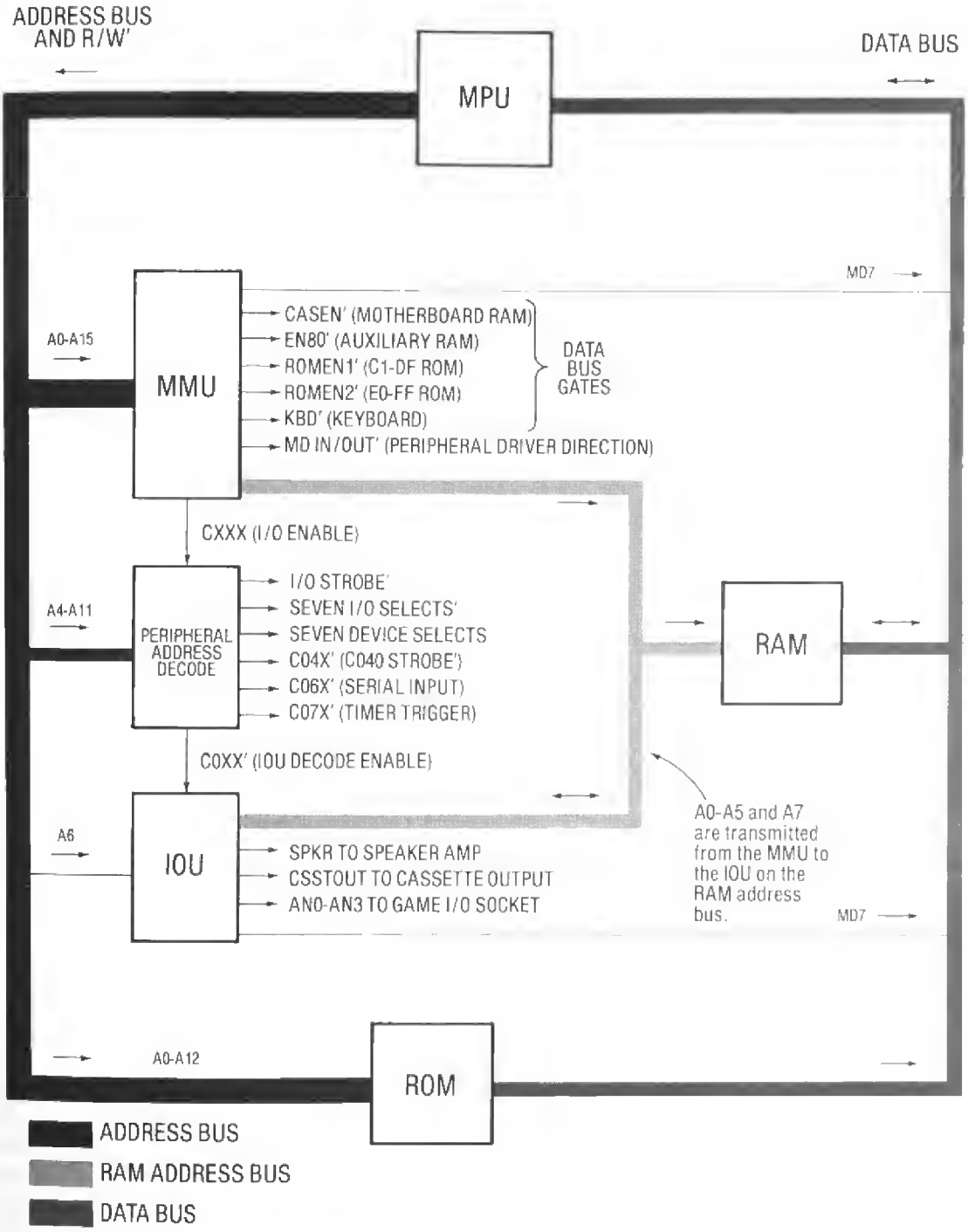


Figure 2.5 Bus Diagram: Address Decoded Signals in the Apple IIe.

the IOU do not have to monitor the full address bus because they receive enabling inputs directly or indirectly from the MMU. The Apple management signals, decoded from the address bus and output from the MMU, are listed here.

SIGNAL	FUNCTION
CASEN′	Enable data transfer between motherboard RAM and MPU
EN80′	Enable data transfer between auxiliary card RAM and MPU
ROMEN1′	Enable C1—DF ROM
ROMEN2′	Enable E0—FF ROM
CXXX	Enable I/O address decoding
KBD′	Enable keyboard
MD IN/OUT′	Control direction of bidirectional peripheral data bus driver

Other address decoding takes place in the MMU which does not directly manipulate these control signals. This includes setting and resetting of memory configuration soft switches and enabling the status of soft switches to MD7 of the data bus for reading by the MPU. For example, $C082 on the address bus is decoded inside the MMU to reset the HRAMRD (high RAM read enable) soft switch. With this soft switch disabled, an MPU read to address $F000 will result in the MPU bringing ROMEN2′ low and subsequent transfer of data from the E0—FF ROM to the data bus. The functional details of the MMU soft switches are not of primary interest here but are a subject of Chapter 5. The important concept here is that the controlling 6502 program manipulates the memory configuration of the Apple by address bus commands decoded in the MMU to set or reset soft switches. Then the MMU, guided by the status of the soft switches, monitors the address bus and enables various functional areas of the Apple via the control signals listed above.

All of the MMU management signals except MD IN/OUT′ and **CXXX** enable the selected device to control the data bus during a read cycle or, in the case of RAM, to receive data from the data bus during a write cycle.* MD IN/OUT′ controls the direction of a bidirectional peripheral data bus driver as described in the next section. CXXX enables further address decoding in the $CXXX range in the **peripheral address decoding circuitry**. The signals output by the peripheral address decoding circuitry are

- an I/O STROBE′ signal to the seven peripheral slots,
- an I/O SELECT′ signal to each of the peripheral slots,
- a DEVICE SELECT′ signal to each of the peripheral slots,
- the C040 STROBE′ output,
- the C06X′ serial input enable signal,
- the C07X′ timer trigger,
- and the C0XX′ signal to the IOU.

The I/O STROBE′, I/O SELECT′, and DEVICE SELECT′ signals are used by the peripheral slots in a variety of ways described in Chapters 6 and 7. In many instances, the effect is to enable data bus communication with a peripheral card. The C040 STROBE′ is a game I/O socket output that goes low for one half of a microsecond when $C04X is on the address bus. C06X′ enables one of eight serial inputs to MD7 of the data bus during a read cycle. C07X′ triggers the four timers whose durations depend on settings of paddles, joysticks or other variable resistors. The **C0XX**′ signal enables further address decoding in the IOU.

Address decoding in the **IOU** is not as extensive as it is in the MMU. The IOU only monitors parts of the $C000—$C05F range to set or reset some video configuration soft switches, to gate the status of various IOU flags and soft swithches to MD7 of the data bus for reading by the MPU, and to directly control some serial outputs. The serial control signals which come from the IOU are

- ANNUNCIATORS 0—3 to the game I/O socket,
- SPKR to the speaker amplifier,
- and CASSO to the cassette output voltage divider.

Figure 2.5 shows that the only line of the address bus connected to the IOU is A6. Even with the aid of the C0XX′ input, the IOU needs more addressing inputs to perform its decoding functions. It needs to monitor A0 to distinguish between a switch on and switch off function. It needs to monitor A3 to distinguish between a video soft switch command and an annunciator command. In fact, to perform all of its decoding functions, the IOU needs to monitor A0—A7 of the 6502 address in addition to monitoring the C0XX′ line. However, with the exception of A6, it

*Some terminology examples—$CXXX is the address range $C000—$CFFF. CXXX is a signal which goes **high** when an address in the $CXXX range is on the address bus. C06X′ is a signal which goes **low** when an address in the $C06X range is on the address bus.

Table 2.1 Apple IIe Master Address Decode Table (1 of 2).

FUNCTION	RW	HEX RANGE	DECIMAL RANGE	DECIMAL COMPLEMENT
RESET/SET 80STORE	W	$C000/$C001	49152/49153	−16384/−16383
RESET/SET RAMRD	W	$C002/$C003	49154/49155	−16382/−16381
RESET/SET RAMWRT	W	$C004/$C005	49156/49157	−16380/−16379
RESET/SET INTCXROM	W	$C006/$C007	49158/49159	−16378/−16377
RESET/SET ALTZP	W	$C008/$C009	49160/49161	−16376/−16375
RESET/SET SLOTC3ROM	W	$C00A/$C00B	49162/49163	−16374/−16373
RESET/SET 80COL	W	$C00C/$C00D	49164/49165	−16372/−16371
RESET/SET ALTCHRSET	W	$C00E/$C00F	49166/49167	−16370/−16369
READ KBD/KEYSTROBE	R	$C00X	49152—49167	−16384 TO −16369
RESET KEYSTROBE	RW	$C010	49168	−16368
RESET KEYSTROBE	W	$C01X	49168—49183	−16368 TO −16353
READ KBD/AKD	R	$C010	49168	−16368
READ KBD/HRAM BANK2	R	$C011	49169	−16367
READ KBD/HRAMRD	R	$C012	49170	−16366
READ KBD/RAMRD	R	$C013	49171	−16365
READ KBD/RAMWRT	R	$C014	49172	−16364
READ KBD/INTCXROM	R	$C015	49173	−16363
READ KBD/ALTZP	R	$C016	49174	−16362
READ KBD/SLOTC3ROM	R	$C017	49175	−16361
READ KBD/80STORE	R	$C018	49176	−16360
READ KBD/VBL′	R	$C019	49177	−16359
READ KBD/TEXT	R	$C01A	49178	−16358
READ KBD/MIXED	R	$C01B	49179	−16357
READ KBD/PAGE2	R	$C01C	49180	−16356
READ KBD/HIRES	R	$C01D	49181	−16355
READ KBD/ALTCHRSET	R	$C01E	49182	−16354
READ KBD/80COL	R	$C01F	49183	−16353
TOGGLE CASSETTE OUT	RW	$C02X	49184—49199	−16352 TO −16337
TOGGLE SPEAKER	RW	$C03X	49200—49215	−16336 TO −16321
C040 STROBE′	RW	$C04X	49216—49231	−16320 TO −16305
RESET/SET TEXT	RW	$C050/$C051	49232/49233	−16304/−16303
RESET/SET MIXED	RW	$C052/$C053	49234/49235	−16302/−16301
RESET/SET PAGE2	RW	$C054/$C055	49236/49237	−16300/−16299
RESET/SET HIRES	RW	$C056/$C057	49238/49239	−16298/−16297
RESET/SET AN0	RW	$C058/$C059	49240/49241	−16296/−16295
RESET/SET AN1	RW	$C05A/$C05B	49242/49243	−16294/−16293
RESET/SET AN2	RW	$C05C/$C05D	49244/49245	−16292/−16291
RESET/SET AN3	RW	$C05E/$C05F	49246/49247	−16290/−16289
READ CASSETTE IN	R	$C060,$C068	49248,49256	−16288,−16280
READ PB0	R	$C061,$C069	49249,49257	−16287,−16279
READ PB1	R	$C062,$C06A	49250,49258	−16286,−16278
READ PB2	R	$C063,$C06B	49251,49259	−16285,−16277
READ TIMER0	R	$C064,$C06C	49252,49260	−16284,−16276
READ TIMER1	R	$C065,$C06D	49253,49261	−16283,−16275
READ TIMER2	R	$C066,$C06E	49254,49262	−16282,−16274
READ TIMER3	R	$C067,$C06F	49255,49263	−16281,−16273
TRIGGER TIMERS	RW	$C07X	49264—49279	−16272 TO −16257

Table 2.1 Apple IIe Master Address Decode Table (2 of 2).

FUNCTION	RW	HEX RANGE	DECIMAL RANGE	DECIMAL COMPLEMENT
HIGH RAM, BANK2				
WCNT = 0,W',R	RW	$C080,$C084	49280,49284	−16256,−16252
WCNT+1,R'	R	$C081,$C085	49281,49285	−16255,−16251
WCNT = 0,R'	W	$C081,$C085	49281,49285	−16255,−16251
WCNT = 0,W',R'	RW	$C082,$C086	49282,49286	−16254,−16250
WCNT+1,R	R	$C083,$C087	49283,49287	−16253,−16249
WCNT = 0,R	W	$C083,$C087	49283,49287	−16253,−16249
HIGH RAM, BANK1				
WCNT = 0,W',R	RW	$C088,$C08C	49288,49292	−16248,−16244
WCNT+1,R'	R	$C089,$C08D	49289,49293	−16247,−16243
WCNT = 0,R'	W	$C089,$C08D	49289,49293	−16247,−16243
WCNT = 0,W',R'	RW	$C08A,$C08E	49290,49294	−16246,−16242
WCNT+1,R	R	$C08B,$C08F	49291,49295	−16245,−16241
WCNT = 0,R	W	$C08B,$C08F	49291,49295	−16245,−16241
DEVICE SELECT' SLOT 1	RW	$C09X	49296—49311	−16240 TO −16225
DEVICE SELECT' SLOT 2	RW	$C0AX	49312—49327	−16224 TO −16209
DEVICE SELECT' SLOT 3	RW	$C0BX	49328—49343	−16208 TO −16193
DEVICE SELECT' SLOT 4	RW	$C0CX	49344—49359	−16192 TO −16177
DEVICE SELECT' SLOT 5	RW	$C0DX	49360—49375	−16176 TO −16161
DEVICE SELECT' SLOT 6	RW	$C0EX	49376—49391	−16160 TO −16145
DEVICE SELECT' SLOT 7	RW	$C0FX	49392—49407	−16144 TO −16129
I/O SELECT' SLOT 1	RW	$C1XX	49408—49663	−16128 TO −15873
I/O SELECT' SLOT 2	RW	$C2XX	49664—49919	−15872 TO −15617
I/O SELECT' SLOT 3	RW	$C3XX	49920—50175	−15616 TO −15361
I/O SELECT' SLOT 4	RW	$C4XX	50176—50431	−15360 TO −15105
I/O SELECT' SLOT 5	RW	$C5XX	50432—50687	−15104 TO −14849
I/O SELECT' SLOT 6	RW	$C6XX	50688—50943	−14848 TO −14593
I/O SELECT' SLOT 7	RW	$C7XX	50944—51199	−14592 TO −14337
I/O STROBE'	RW	$C800—$CFFF	51200—53247	−14336 TO −12289
SET INTC8ROM	RW	$C3XX (INTC3)	49920—50175	−15616 TO −15361
RESET INTC8ROM	RW	$CFFF	53247	−12289
LOWER 48 RAM ACCESS	RW	$0000—$BFFF	00000—49151	−65536 TO −16385
HIGH RAM ACCESS	RW	$D000—$FFFF	53248—65535	−12288 TO −00001
INT/SLOT ROM ACCESS	RW	$C100—$CFFF	49408—53247	−16128 TO −12289
HIGH ROM ACCESS	R	$D000—$FFFF	53248—65535	−12288 TO −00001

does not receive these low order address inputs directly from the address bus. It receives them from the MMU via the multiplexed RAM address bus as described in the previous section.

The control functions of various addresses are fundamental operational features of the Apple IIe computer. For easy reference, Table 2.1 contains a complete list of the address decoded functions of the Apple IIe.

I/O (INPUT/OUTPUT)

The I/O capability of the Apple IIe is as versatile as microcomputer bus architecture. We have seen how the video scanner shares RAM, the RAM address bus, and the data bus to drive a video map out of RAM for video generator processing. The other I/O features require more direct manipulation from the MPU.

Apple I/O is memory mapped. This computer lingo is used to describe a system where the I/O devices have assigned addresses just like memory. The addresses assigned to I/O in the Apple are in the $CXXX range. This includes the built-in I/O devices as well as the peripheral slots.

Figure 2.6 is a bus diagram of the Apple IIe highlighting I/O capabilities. As you would suspect in a memory mapped I/O system, the address bus is

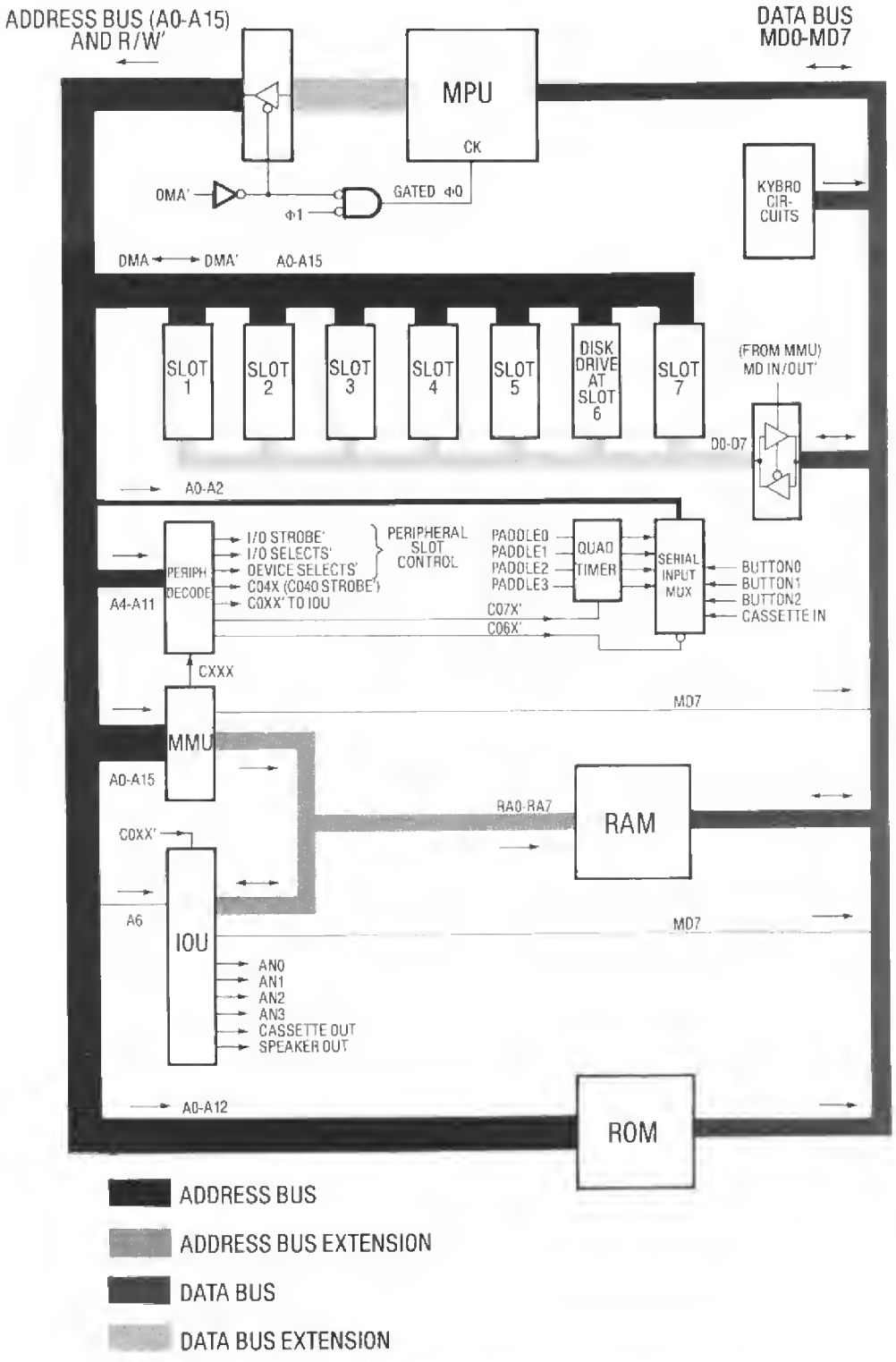


Figure 2.6 Bus Diagram: Input/Output in the Apple IIe.

directly or indirectly distributed to all of the I/O devices. Additionally, most of the I/O devices are connected to the data bus.

Hardware control of the I/O devices is via address decoding. In other words, when the MPU addresses an I/O device, circuitry on the motherboard must detect that address on the address bus and generate signals which control that device.

The response of a device to its control signals will depend on the nature of the device. Addressing the speaker makes the speaker diaphragm tense or relax. Addressing the cassette output causes the cassette output line to toggle high or low. Addressing the keyboard causes the ASCII of the last keypress to be placed on the data bus. Addressing a peripheral slot causes the card in the slot to do whatever it was designed to do when its control signals are activated.

Keyboard Input

The Apple IIe keyboard circuits include the keyboard, a keyboard encoder IC, and a 2K x 8 ROM. The keyboard and encoder combine to latch ASCII for keys that are pressed. The 2K x 8 ROM gives the Apple IIe a versatile keyboard code translation capability and provides a tri-state connection to MD0—MD6 of the data bus. Since the keyboard code is latched, the controlling program can make the MPU read the code of the last keypress at any time or any number of times before the following keypress.

The MPU reads the keyboard input via a read access to $C000. Any read access in the $C00X range can be used for this purpose, but the programming convention is to use $C000. When the MMU detects a read to $C00X on the address bus, it pulls the enabling KBD' signal low*. This results in the transfer of the 7-bit ASCII of the last keypress from the keyboard ROM to MD0—MD6 of the data bus. Additionally, the IOU detects the read to $C00X and places the state of its **KEYSTROBE** soft switch on MD7 of the data bus. The MPU thus reads the state of KEYSTROBE and the latched ASCII of the last keypress with a single access to $C00X.

The KEYSTROBE soft switch is set by the KSTRB signal which goes high momentarily any time a matrix key is pressed. KSTRB is output by the keyboard encoder and processed inside the IOU. The strobe soft switch is reset when the MPU makes a read access to $C010 or a write access to $C01X. This provides programmers with a means of detecting a keypress and distinguishing between multiple keypresses. The program polls $C000 until it finds the MSB high (KEYSTROBE). Then it resets KEYSTROBE, processes the ASCII, then resumes polling $C000.

If a key is held down continuously for .5 to .8 seconds (32 to 48 television scans), the IOU will start setting the KEYSTROBE soft switch 15 times every second (once every four television scans). To the program, this looks as if someone is pressing a key 15 times per second, and the result is the auto-repeat feature of the keyboard.

A second flag related to the keyboard is the **AKD** (any key down) flag, read at $C010. The AKD signal is routed from the keyboard encoder to the IOU and relayed to MD7 when the IOU detects a read to $C010. This gives programmers a little more versatility in interpreting keypresses. Note that reading the AKD flag also resets the KEYSTROBE soft switch.

Peripheral Slots

The seven peripheral slots are connected to all of the lines of the address bus and, through a bidirectional driver, to all of the lines on the data bus. The primary purpose of the driver is current amplification. In other words, the driver helps motherboard data bus signal suppliers in driving peripheral card signal receivers and vice versa. Timing and control signals to the driver are such that it doesn't isolate the peripheral slots from data bus signals... with one exception. The driver does prevent video data from motherboard RAM from reaching the peripheral slots during MPU write cycles. This seems to have been done for compatilbility with the Apple II and II Plus. I can see no other reason to deny video data to the peripheral slots during write cycles.

The MMU controls the direction of the peripheral data bus bidirectional driver via the MD IN/OUT' control line. The state of the address bus, R/W', and the DMA' and INHIBIT' lines are used to determine the correct direction for the driver. Direction is in to the data bus when MD IN/OUT' is high and out from the data bus when MD IN/OUT' is low.

I/O SELECT' ($C100—$C7FF), DEVICE SELECT' ($C090—$C0FF), and I/O STROBE' ($C800—$CFFF) signals decoded on the motherboard inform a peripheral card when it is being

*KBD' also goes low when a read is made to $C01X although Apple does not document this feature. Daring programmers may exploit this capability to read the keyboard ASCII simultaneously with AKD or other IOU or MMU flags. Before you write routines like this, please note that AKD becomes valid before keyboard ASCII as described in Chapter 7.

accessed at one of its assigned addresses. But the slots are not restricted to response to $C090—$CFFF addressing. The INHIBIT' line allows any slot to disable motherboard and auxiliary slot response to $0000—$BFFF and $C100—$FFFF addressing. With full connection to the address bus and data bus, peripheral cards can take advantage of this capability in any number of ways.

The peripheral slot and auxiliary slot connections are different from each other in nature. The auxiliary slot is integrated into the Apple's timing and control scheme as an 80-column video card, and connection to the multiplexed RAM address bus, data bus, and video data bus make the auxiliary slot ideal for expansion RAM and 80-column cards. Other connections make the auxiliary slot an ideal diagnostic port for production testing or fault isolation in malfunctioning motherboards. The peripheral slots, on the other hand, are meant to hold any variety of I/O, memory expansion, or system controlling device. To this end, the peripheral slots are supported by full connection to the address bus and data bus, fixed address assignments, and connection to 6502 control lines and Apple timing signal lines.

Disk I/O

Disk I/O operations are an example of the flexibility that the peripheral slots give to the Apple. With no peripheral cards plugged in, the Apple IIe has only an anitiquated cassette interface for loading and saving memory data. This goes back to the bad old days when built-in cassette I/O was a noteworthy convenience. But everybody knows that the primary means of loading and saving memory data in the Apple IIe is with 5 ¼ inch floppy disks. The Apple is thought of as a disk based computer, and when a disk controller is installed in a peripheral slot, it is fully integrated into the Apple, just as if it were a motherboard device.

The data transfer path for disk output is from RAM to the MPU to the disk controller to the disk drive, and the data input path is the reverse of the output path. Data is loaded from the transfer source into the MPU, then stored at the transfer destination from the MPU. Data transfer between the MPU and the controller is via the data bus.

The disk controller resides in a peripheral slot and responds to the address bus/data bus environment much like RAM. During disk input, the controller responds to a read access from the MPU by placing a byte of data on the data bus. During disk output, the controller responds to a write access by accepting a byte of data from the data bus. The addresses of the input port and output port depend on which slot the disk controller is in. If, as is normally the case, the disk controller is in Slot 6, the input port address is $C0EC and the output port address is $C0ED. Besides $C0EC and $C0ED, other address commands perform the functions of motor control, drive selection, read/write configuration, and head positioning. These commands are decoded on the motherboard and controller. The motherboard circuits detect the $C0EX range on the address bus and activate the Slot 6 DEVICE SELECT' signal to tell Slot 6 it is being accessed. The controller decodes A0—A3 of the address bus to determine which of 16 possible commands it is being given.

The actual programming of disk I/O is very complex, requiring timed intervals, data encoding, and extensive software housekeeping. Regardless of this, all MPU control of the disk is via 16 address commands on the address bus, and all data transfer is over the data bus.

There is no motherboard ROM routine to load programs from a disk drive when the Apple is first turned on. A 256-byte program does exist on the controller card, accessible at addresses $C600—$C6FF (assuming Slot 6), which loads the extensive Disk Operating System (DOS) from disk to RAM. After power up, the motherboard firmware turns control over to this controller firmware to get the DOS up and running.

DMA and the MPU

As Figure 2.6 shows, the MPU address and R/W' lines are connected to the address bus via a 17-bit tri-state line driver. One purpose for this device is to enable the MPU to **drive** (supply required signal voltages to) all the circuits on the address bus, including a possible variety of peripheral cards. A second purpose of the address driver is to give the MPU a tri-state connection to the address bus. This is necessary to isolate the MPU from the address bus during **DMA** operation, because the 6502 address and R/W' outputs are not tri-state. DMA (**Direct Memory Access**) is achieved from a peripheral card when the card pulls the DMA' line low. This DMA capability is actually a **direct bus access** which gives the peripheral card command of the entire Apple. Pulling the DMA' line low forces the 17-bit line driver to high impedance, stops the clock to the MPU, forces the MPU data terminals to input mode, and affects the MMU read/write control of the peripheral data bus driver.

Unless it is stated otherwise, the discussions in *Understanding the Apple IIe* assume that no peripheral card is performing DMA. This means that the normal situation exists in which the MPU controls the data bus during write cycles and always controls the address bus.

The Serial Input Multiplexor

In addition to the keyboard input and any peripheral card inputs to the Apple, there are four paddle inputs, three pushbutton inputs, and the cassette input. Each paddle input is tied to a timer which, when triggered, outputs a high TTL level for a period of time determined by its paddle setting. The four timer pulses, three pushbutton inputs, and the processed cassette input are all applied to the **serial input multiplexor**.

When an address in the $C06X range is on the address bus, the serial input multiplexor places one of its eight inputs on D7 of the peripheral data bus as follows:

ADDRESS	INPUT
$C060/$C068	Cassette input
$C061/$C069	Pushbutton 0
$C062/$C06A	Pushbutton 1
$C063/$C06B	Pushbutton 2
$C064/$C06C	Timer 0
$C065/$C06D	Timer 1
$C066/$C06E	Timer 2
$C067/$C06F	Timer 3

The MMU brings MD IN/OUT' high when a read is made in the $C06X range. This causes the serial input data to be passed from D7 of the peripheral bus through the bidirectional driver to MD7 of the data bus. The combined response of the serial input multiplexor and the bidirectional driver to a read to $C06X allows the MPU to read the serial inputs like memory.

The serial input mechanization is similar to ROM. A device responds to its address on the address bus by placing data on the data bus. In this case, however, data is placed on only one line of the data bus. The MPU receives data from the data bus as it does when reading data from memory, and the controlling program ignores everything but MD7. The program processes the MD7 information, extracts the transfer data, and stores it in RAM.

The Serial Outputs

In addition to the video output and any peripheral card outputs, there are seven serial outputs from the Apple motherboard. These outputs are operated by address decoding. They are direct or indirect outputs of the IOU, with the exception of the C040 STROBE' which is an output of the peripheral address decoding circuitry. The serial outputs and their controlling addresses are

CONTROL ADDRESS	SERIAL OUTPUT
$C02X	Cassette output toggle
$C03X	Speaker toggle
$C04X	C040 STROBE'
$C058/$C059	ANNUNCIATOR 0 off/on
$C05A/$C05B	ANNUNCIATOR 1 off/on
$C05C/$C05D	ANNUNCIATOR 2 off/on
$C05E/$C05F	ANNUNCIATOR 3 off/on

A very interesting point about the serial outputs is that serial output data is not transferred on the data bus. Most of us would expect a serial output to be written out on one of the lines of the data bus as if we were writing to memory. But addressing a serial output port merely performs a control function on the output line. For example, addressing the cassette output port toggles the cassette output line, meaning it changes the high/low state of the output line to the opposite state. In other words, the programmer does not write data to the cassette by sending data over the data bus to an output line. Instead, he either tells the line to change states or refrains from telling the line to change states at a timed interval.

Other serial output is similar to the cassette output. The output port is addressed, and the control function—toggle, strobe, level high, or level low—is performed. Speaker, annunciator, and C040 STROBE' output lines are controlled directly by address decode in a process which ignores the data bus. The speaker is a toggle output like the cassette output. The programmer can toggle the high/low state, but he never knows whether the state is high or low. The annunciators are on/off outputs which can be brought high or low. For example, $C058 makes ANNUNCIATOR 0 go low, and $C059 makes ANNUNCIATOR 0 go high. The C040 STROBE' simply goes low for half a microsecond any time $C04X is on the address bus, then returns high.

Reading or writing to a serial output port is a control access as opposed to a data access. The MPU reads from the data bus or writes to it on every 6502 cycle, even in a control access. The programmer performs a control access with a normal read or write instruction, but the data that is read or written is irrelevant and ignored. This is why statements like "SPEAKER=PEEK(-16336)" are made in BASIC to control the speaker and the data is

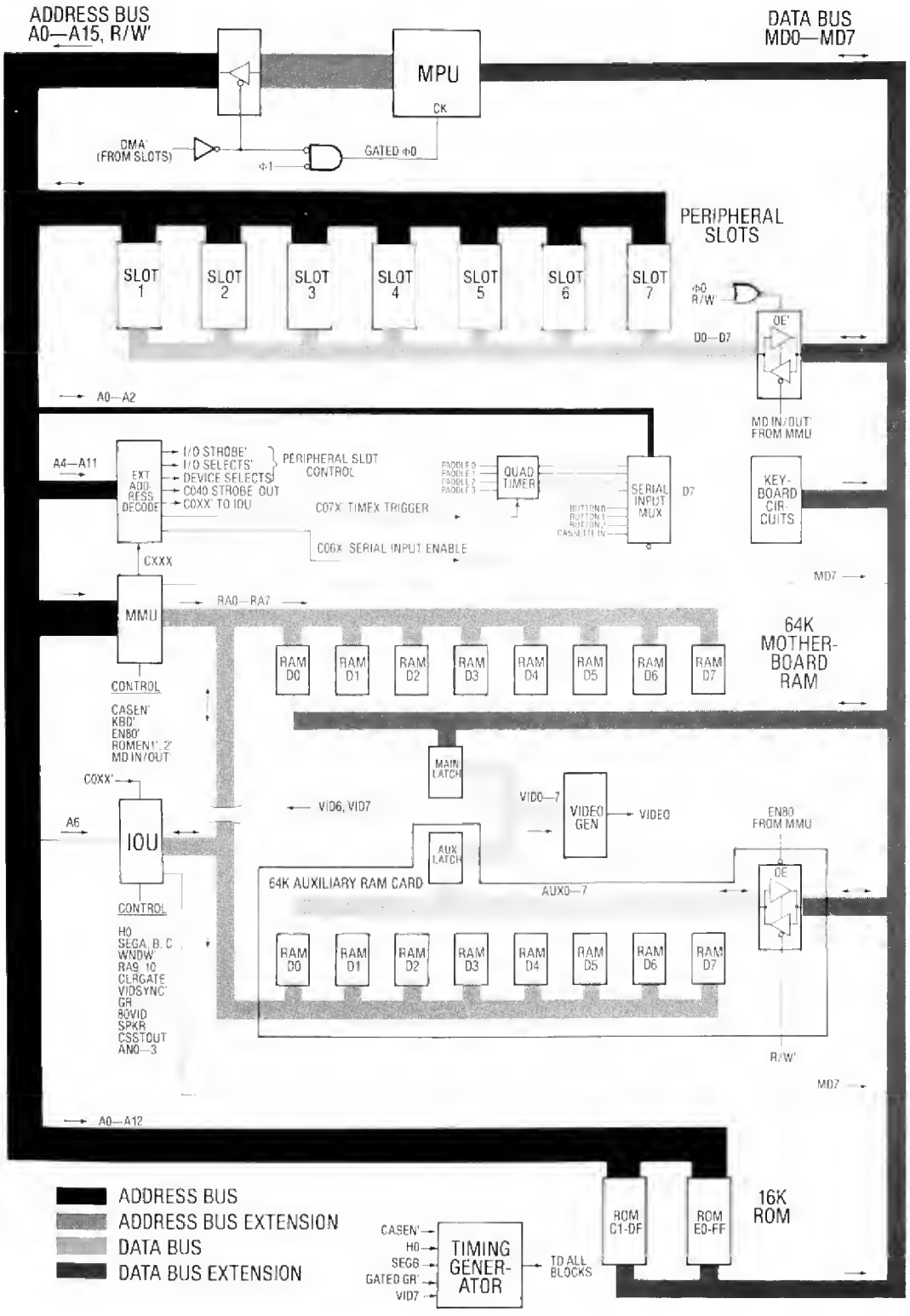


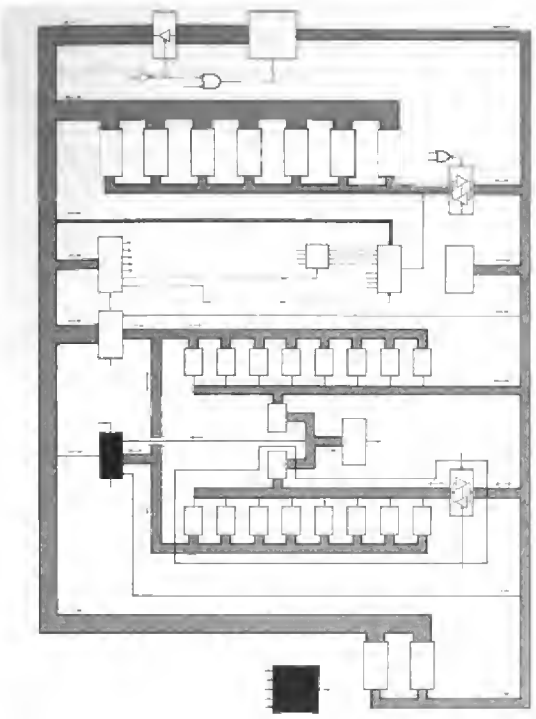
Figure 2.7 The Apple IIe Bus Structure.

ignored. The programmer is making a control access to -16336 ($C030, the speaker port), and the data is irrelevant.

THE COMPLETED BUS STRUCTURE

The discussion of the bus structure of the Apple IIe is now complete. This chapter has presented a series of diagrams of the bus structure, building in complexity and completeness as we progressed from basic ideas to detailed structure. Figure 2.7 is the final diagram in this series. The author feels that study of this diagram is very important in the effort to understand the Apple IIe computer. It is hoped that the reader can become comfortable with the concepts of information flow within the bus structure, because this chapter is the foundation upon which all that follows is built.

The remaining chapters are devoted to a more detailed discussion of the various functional areas of the Apple IIe, beginning with the important subject of timing. Understanding these detailed discussions will be much easier if the reader attempts to visualize how each area performs its functions within the bus structure.



chapter 3

Timing Generation and the Video Scanner

Most operational aspects of the Apple IIe have now been discussed within the context of the bus structure. However, this discussion has left out one of the Apple's most important operational aspects—timing. Timing synchronizes everything that goes on in the Apple. To discuss it, we must get into real nuts and bolts detail about computer operation.

Up to this point, the subject matter of *Understanding the Apple IIe* has been of a general nature. No attempt was made in Chapters 1 and 2 to explain the finer points of Apple IIe operation. Having gained understanding of the Apple's bus structure, you are largely aware of the methods of communication and control that take place in this computer. The following chapters will build on this foundation of understanding, examining and discussing the detailed features of all functional areas of the Apple IIe.

The perceptive reader is probably getting the message that the going is about to become stickier. This book attempts to explain as much as possible about the operation of the Apple in understandable English. There comes a point, however, beyond which clear illustration is achieved only with such technical tools as timing diagrams, truth tables, logic diagrams and schematic diagrams. One of the goals of *Understanding the Apple IIe* is to assist those readers who desire to do so to analyze the operation of the Apple IIe in depth. For this reason, some technically oriented analysis aids are presented in this chapter and succeeding chapters. These technical aids will be accompanied by technical language. Every person reading these words is capable of understanding the technical sections, but some readers may not wish to, and others will find it a struggle. Every effort has been made to assist all readers in achieving fullest possible understanding from the least possible effort.

By way of warning, the details of some functional areas are just plain difficult, but most of the areas are pretty painless.* In particular, much of the complexity of the Apple is concentrated in RAM and its associated circuitry. Some other complicated

*Even though it is not part of the motherboard circuitry, disk I/O is the subject of a chapter of *Understanding the Apple IIe*. Readers intrepid enough to tackle this chapter will find disk I/O to be a complex but interesting area of study.

circuitry, like the internal workings of the MPU, will not be discussed at all. Besides the RAM circuitry, the most difficult topics probably are the details of timing and video generation. Timing comes next, so put on your overshoes—we're going wading.

TIMING OVERVIEW

The important timing signals in the Apple IIe all originate at a small group of circuits called the **timing generator**. You should appreciate this when studying the Apple, because it makes a difficult job easier. Interrelated digital timing originating from multiple sources can scramble your brains. With a single timing source we can assimilate the timing sequences and then apply them to the various functional areas in the following chapters.

Timing signals are distributed to all areas of the Apple, but the Apple's timing requirements are determined primarily by RAM usage. RAM is accessed alternately by the 6502 processor and the video scanner. Executing a stored sequential program and generating a color television video signal are two entirely different tasks, but the two tasks are synchronized in the Apple. As we shall see, execution of this double task dictates certain facts of life about Apple timing.

The timing generator controls the timing and affects all areas of the Apple IIe. Some of these areas also affect timing generation (see Figure 3.1). The external influences are as follows:

1. One of the timing signals, CAS', is enabled or disabled by CASEN' from the MMU.
2. VID7 of the video data bus and the display mode affect the generation of the LDPS' and VID7M video timing signals.
3. An auxiliary card working in coordination with a Slot 1 peripheral card can disable all of the timing signals and substitute alternate signals. This is not normally done in operational Apples, but it is a capability.
4. Feedback from the video scanner elongates one system clock period toward the end of each horizontal television scan.

The elongation referred to in item 4 above is necessary to keep colors consistent from scan to scan. It also means the clock period of the 6502 is not constant but is elongated on every 65th cycle. This book will refer to this elongated machine cycle as the **long cycle**. Because of the feedback from the video scanner to the timing generator, the two areas are covered in this single chapter.

Apple timing originates with a 14.31818 MHz crystal oscillator. The output of the oscillator, referred to as 14M, is a voltage which switches from low to high and back very close to 14,318,180 times every second. The reason for using 14.31818 MHz instead of 14 MHz is that 14,318,180 Hz divided by four is 3,579,545 Hz, the exact frequency at which color information is passed in a television set. All of the distributed timing signals are clocked by low to high transitions of the 14M clock, so the exact frequencies at which events occur in the Apple are determined by a television signal specification. The approximate frequencies at which some functions occur are:

FUNCTION	APPROXIMATE FREQUENCY
6502 Cycle	1 MHz
Video Scanner Increment	1 MHz
Address Bus Access	1 MHz
RAM Access	2 MHz
COLOR REFERENCE	3.5 MHz
Video Output	7 MHz max.

All of these frequencies are determined by outputs of the timing generator.

The timing generator circuits consist of a 14.31818 MHz oscillator, a pair of divide-by-two flip-flops, and a **HAL** (Hard Array Logic) IC. The HAL is a special type of IC whose logic functions can be programmed within the constraints of a format. The format of the Apple IIe timing HAL is a 20-pin IC with eight registered (clocked) outputs driven by eight external inputs. This HAL, programmed to Apple's specifications, performs much of the work in generating timing signals for the Apple IIe.

THE TIMING SIGNALS

This section is a very brief description of the timing signals which are the outputs of the timing generator. All these signals are described in detail later in this chapter.

PHASE 0 is the 1 MHz clock input to the 6502. It also is used as a general timing reference in the MMU and IOU and throughout the motherboard. PHASE 0 defines when an MPU address is valid, and whether the MPU or the video scanner is addressing RAM. It is available at the peripheral slots.

PHASE 1 is PHASE 0 inverted or PHASE 0'. It is inverted and gated by DMA' to provide the 1 MHz clock input to the 6502. PHASE 1 is used as a timing reference by several motherboard devices and is also available at the peripheral slots.

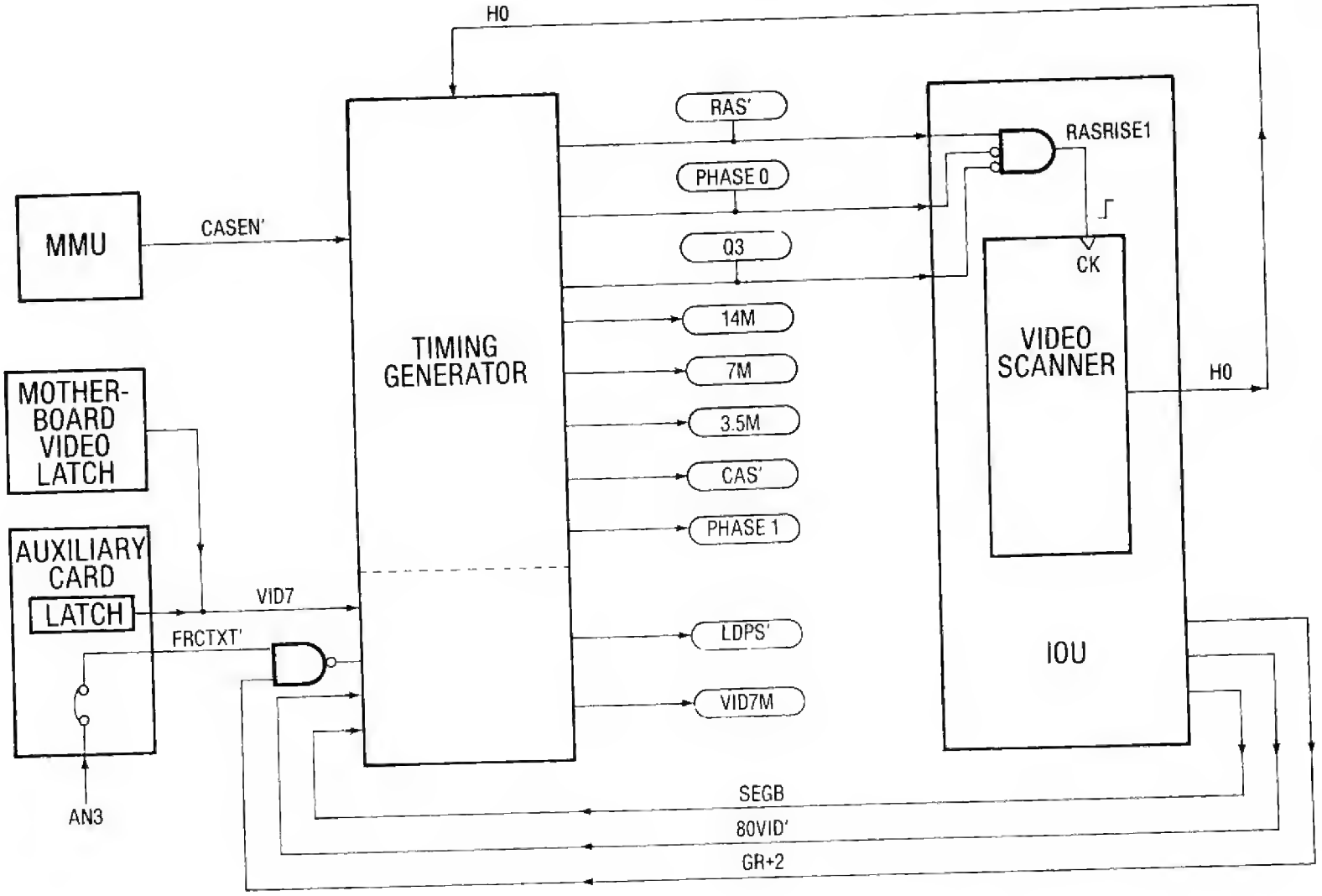


Figure 3.1 Functional Diagram: The Timing Generator and Video Scanner.

COLOR REFERENCE is a 3.5 MHz clockpulse which is used to make up the color burst portion of the video output. The color of any Apple video is determined by its phase relationship with the COLOR REFERENCE signal. COLOR REFERENCE is available at peripheral Slot 7.

7M is a 7 MHz clock used only in the generation of other timing signals. It is also available at the peripheral slots.

14M is the output of the Apple's 14 MHz clockpulse oscillator. It is used in timing generation and in the shifting of video patterns in the video generator. As mentioned in the timing overview section, 14M is the ultimate source of Apple timing.

RAS' (Row Address Strobe) clocks ROW address information to RAM, and serves as a timing reference in the IOU and MMU. Among other things, RAS' defines RAM ROW address time, and RAS' rising during PHASE 1 causes the video scanner to increment. RAS' occurs twice every 6502 cycle—once for MPU access and once for video scanner access.

CAS' (Column Address Strobe) clocks COLUMN address information to motherboard RAM. CAS' is gated by CASEN' from the MMU during PHASE 0 to enable or disable motherboard RAM. CAS' always falls during PHASE 1 and falls during PHASE 0 if CASEN' is low.

Q3 is a 2 MHz signal used as a timing reference in the MMU and IOU. It is also available at the peripheral slots.

LDPS' (LoaD Parallel in/Serial out register) is a video timing term that defines a video cycle. Picture patterns are loaded while LDPS' is low and shifted out to the VIDEO output line when LDPS' is high. LDPS' occurs once every 6502 cycle in SINGLE-RES display modes and twice every 6502 cycle in DOUBLE-RES display modes.

VID7M is a video timing signal that enables the 14M clockpulse of the video shift register. It enables shifting every other 14M in TEXT40 and HIRES40 display modes and shifting every 14M in the other display modes. It also may be delayed or undelayed in HIRES GRAPHICS mode to control the shifting of 7-dot groups.

APPLE FREQUENCIES

It is very hard to make precise statements about the frequencies of some signals in the Apple. This is because of the clockpulse elongation which occurs every 65th 6502 cycle. 14M, 7M, and COLOR REFERENCE are not affected by this elongation. PHASE 0, PHASE 1, Q3, RAS', and CAS' are affected.

If not for the long cycle, the frequencies of all timing signals could be computed by dividing 14,318,180 by 14, 7, 4, 2, or 1. In actuality, this works for computing the fixed frequencies. 14M occurs at 14.31818 MHz; 7M occurs at 7.15909 MHz; COLOR REFERENCE occurs at 3.579545 MHz. The 1 MHz and 2 MHz signals are less straightforward.

The period of time required for a 14.31818 MHz signal to go through a complete high/low cycle is 1/14318180 seconds or about 69.8 nanoseconds (69.8 billionths of a second). All synchronized durations in the timing generator are multiples of this time period which we will call the PERIOD for this discussion.

The normal 6502 machine cycle lasts 14 PERIODS or about .978 microseconds. The long cycle lasts 16 PERIODS or about 1.12 microseconds. There are three frequencies involved here: the primary frequency at which the 6502 is operated for 64 out of 65 cycles, 1.0227 MHz; the secondary frequency at which the 6502 operates for 1 out of 65 cycles, .8949 MHz; and the composite frequency which actually is the number of machine cycles per second, 1.0205 MHz.

The 2 MHz signals are similar to PHASE 0 except that only one of every 130 cycles is elongated. Their normal duration is seven PERIODS or about .489 microseconds. Their long duration is nine PERIODS or about .629 microseconds.

The durations and frequencies of the signals of the timing generator are shown in Table 3.1 below. The values are arithmetic derivations of 14.31818, carried to ten place accuracy. Actual frequencies will vary as the 14M oscillator varies from 14,318,180 Hz due to thermal environment and crystal tolerance.

Also shown in Table 3.1 are correction factors for the 50 Hz IOUs and 14.25 MHz oscillators found on Apple IIe **PAL** (Phase Alternating Lines) motherboards. PAL motherboards are designed for countries using a 50 Hz television scan instead of the American 60 Hz scan. Those PAL motherboards with discrete circuit 14M oscillators use a 14.25045 MHz crystal instead of 14.31818, and those with hybrid circuit 14M oscillators operate at 14.25 MHz. The reason for the different frequency is so the 50 Hz Apple IIe horizontal television scan will approximate the 50 Hz standard of 64 microseconds. As a side effect, 50 Hz Apple execution speed is slightly slower than 60 Hz Apple execution speed.

Table 3.1 Durations and Frequencies of Timing Signals.

SIGNAL	NORMAL DURATION (nsec)	LONG DURATION (nsec)	AVERAGE DURATION (nsec)	PRIMARY FREQUENCY (MHz)	SECONDARY FREQUENCY (MHz)	COMPOSITE FREQUENCY (MHz)
PHASE 0	977.7779019	1117.460459	979.9268644	1.022727143	.89488625	1.02048432
RAS',CAS',Q3	488.888951	628.5715084	489.9634322	2.045454286	1.590908889	2.04096864
COLOR REF	279.3651148			3.579545		
7M	139.6825574			7.15909		
14M	69.84127871			14.31818		

SCAN NOTES:

There are 912 14M periods in a horizontal scan.
There are 262 horizontal scans in a 60 Hz vertical scan.
There are 312 horizontal scans in a 50 Hz vertical scan.

PAL NOTES:

Multiply NTSC frequency by .9952696502 for discrete circuit PAL frequency.
Multiply NTSC duration by 1.004752832 for discrete circuit PAL duration.
Multiply NTSC frequency by .9952382216 for hybrid circuit PAL frequency.
Multiply NTSC duration by 1.004784561 for hybrid circuit PAL duration.

More information on export Apples is given later in this chapter and in Chapter 8. However, much of the discussion in this book assumes we are talking about American Apples. This is particularly true of topics mentioning frequencies or durations or details of television scanning and video generation. Owners of PALbased Apple IIe's should read the section on export Apples in Chapter 8 to get an good idea of the areas of difference between 50 Hz and 60 Hz Apple IIe's.

It is reasonable to wonder why the exact frequencies in the Apple should be of any concern. In fact, for most purposes, the exact frequencies are not important. They are important when discussing television compatibility, because television signals require some specific frequencies which are not exact multiples of 1 MHz. Frequency is also important in so far as it affects MPU execution speeds. Knowledge of 6502 clock speed is very important for Apple programs with precision timing loops. For the most part, we will continue to refer to frequencies in very rough estimates such as 1 MHz or 3.5 MHz.

THE TIMING DIAGRAM

Timing is usually summarized in **timing diagrams**. Figure 3.2 is a timing diagram showing the outputs of the timing generator and some related signals. The timing diagram is a series of line graphs of voltage as a function of time. Voltage changes vertically in the diagram as time passes from left to right.

In the following discussions of timing signals, the reader is encouraged to refer to Figure 3.2 as necessary to clarify relationships in his own mind. Time periods will be measured in millionths of a second (microseconds) and billionths of a second (nanoseconds).

Figure 3.2 shows three 6502 machine cycles—two normal length cycles and one long cycle. For each normal machine cycle, there is one PHASE 0 cycle, two RAS', CAS', and Q3 cycles, three and a half COLOR REFERENCE cycles, seven 7M cycles and fourteen 14M cycles. For reference, the period of 14M is about 70 nanoseconds and the period of a normal PHASE 0 cycle is about 978 nanoseconds.

The signals illustrated in Figure 3.2 are the timing generator outputs, plus AX, H0, and VID7. AX (Address Multiplex) is a signal which is used only inside the timing HAL. It was used in the Apple II to gate ROW or COLUMN addressing to the multiplexed RAM address bus. It can be viewed in the Apple IIe at pin 18 of the HAL.

H0, the least significant bit of the video scanner, is an output of the IOU and an input to the HAL. Its level alternates approximately when PHASE 0 rises for 64 out of 65 MPU cycles. Every 65th cycle,

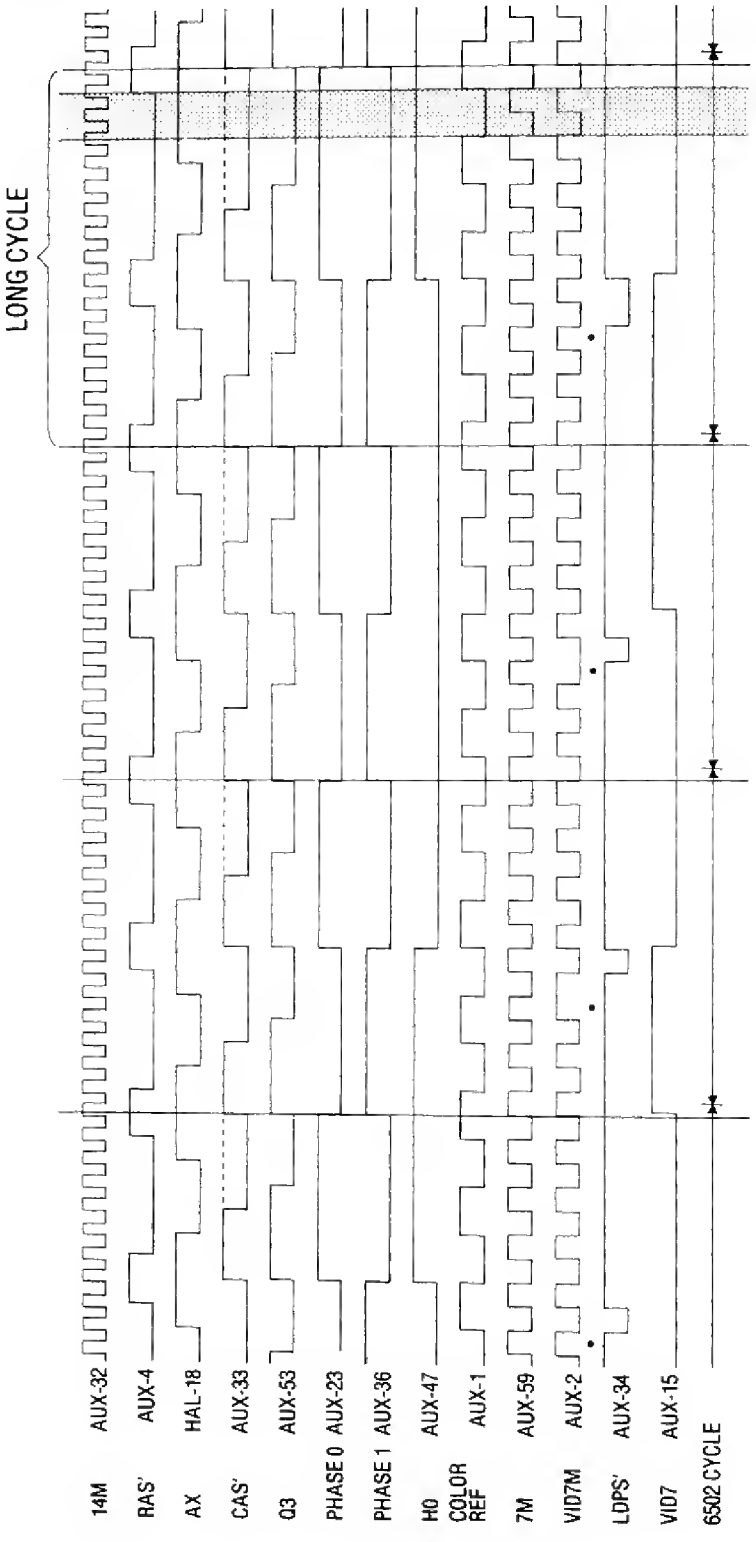


Figure 3.2 Timing Diagram: Timing Generator Signals.

though, H0 stays low for one extra period. It will shortly be shown that there are 65 cycles in a horizontal video scan line. The double period with H0 low, shown in the middle of Figure 3.2, occurs at the right edge of the Apple display window.

Look, for a moment, at PHASE 0 at the left side of Figure 3.2. When PHASE 0 falls for the first time, COLOR REFERENCE is high, but when PHASE 0 falls at the end of the next cycle, COLOR REFERENCE is low. This alternating relationship between PHASE 0 and COLOR REFERENCE is a consequence of the fact that there are 3.5 cycles of COLOR REFERENCE in one cycle of PHASE 0. The relationship can be defined in terms of H0. COLOR REFERENCE is low when PHASE 0 falls during H0'. COLOR REFERENCE is high when PHASE 0 falls during H0.

The H0/PHASE 0/COLOR REFERENCE phase relationship is as described above for 64 out of 65 PHASE 0 cycles. It must be this way during video display periods to conform to the overall scheme for controlling the colors in the GRAPHICS mode of the Apple. The relationship is thrown off, however, by the fact that there are an odd number of PHASE 0 cycles (65) in a horizontal scan. If this were not corrected for, the relationship would reverse every horizontal scan.

The correction occurs after the double period with H0 low. The HAL, monitoring H0 and the timing signals, detects the fact that the relationship has changed. It corrects the relationship by delaying generation of the 1 MHz and 2 MHz signals for one half of a COLOR REFERENCE period. The delay extends the high duration of PHASE 0 and AX, and it extends the low duration of RAS', CAS', and Q3. It also causes the extension of the current 6502 machine cycle (the long cycle). The point at which this delay takes place is shaded in Figure 3.2.

With the exception of LDPS' and VID7M, the timing signals remain fixed in the cyclic patterns illustrated in Figure 3.2. LDPS' and VID7M will vary with the Apple display mode and, in HIRES40 mode, with VID7 of the video data bus. The video timing shown in Figure 3.2 is for HIRES40 mode, and VID7M and LDPS' are shown reacting to VID7. Chapter 8 contains timing diagrams showing other variations of LDPS' and VID7M.

The signals of Figure 3.2 do not actually rise and fall instantly. It takes them about six nanoseconds to rise and fall. Also, it takes a small amount of time for the outputs of an IC to respond to changes in its inputs. The delay from input change to output response is referred to as **propagation delay**.

It is very difficult to illustrate minute propagation delay in a diagram with the time scale of Figure 3.2. Figure 3.3 more accurately depicts the delay hierarchy that exists. The rising edge of 14M is the master reference of Apple timing, and the basic features of propagation delay are:

1. RAS', CAS', Q3, PHASE 0, PHASE 1, LDPS', VID7M, 7M, and COLOR REFERENCE are all clocked by the rising edge of 14M. COLOR REFERENCE and 7M are outputs of a 74S109 with a delay of roughly nine nanoseconds from 14M rising. RAS', CAS', Q3, PHASE 0, PHASE 1, LDPS', and VID7M are outputs of the 16R8 HAL with a delay of roughly 14 nanoseconds from 14M rising.
2. PHASE 0 is routed to the 6502 through one logic device. Internal 6502 actions cause a further delay before the 6502 data clock (the falling edge of the 6502 PHASE 2 clock). The typical 6502 internal delay is not specified in data sheets. The delay between PHASE 0 falling at the peripheral slots and PHASE 2 falling at the 6502 was measured by the author at 28 nanoseconds.* This delay should vary considerably from 6502 to 6502.
3. The video scanner in the IOU is clocked by the rising edge of RAS' during PHASE 1. The delay between the rising edge of 14M and a change in H0 was measured by the author at 80 nanoseconds.** In other words, H0 changes at approximately the same time PHASE 0 rises. This delay should vary considerably from IOU to IOU.

TIMING SIGNAL DISTRIBUTION

Figure 3.4 shows the distribution of timing generator outputs throughout the motherboard. Each motherboard device receives the signals it requires to stay synchronized with the overall Apple timing scheme. Note that all timing generator outputs are connected to pins of the auxiliary slot, but only PHASE 0, PHASE 1, Q3, 7M, and COLOR REFERENCE are available at the peripheral slots. Peripheral cards must perform their functions without the benefits of monitoring 14M, RAS', CAS', LDPS', and VID7M.

*Synertek SY6502 (marking 8307, S10891, 370-6502) in a Revision B Apple IIe computer.

**AMI IOU (marking 8307 MAA, 344-0020-A) in a Revision B Apple IIe computer.

DETAILED DESCRIPTION OF THE TIMING SIGNALS

The following sections describe in detail how the timing signals of the Apple IIe are used. Please refer to Figures 3.2 (timing diagram) and 3.4 (timing signal distribution) as needed while reading these discussions.

PHASE 0 and PHASE 1

PHASE 0 and PHASE 1 provide the primary 1 MHz timing reference of the Apple IIe computer. They could easily (and more properly) be called 1M and 1M′ to avoid confusion with the 6502 PHASE 0 clock input and 6502 PHASE 1 internal clock. As the names 1M and 1M′ imply, PHASE 1 is simply the exact inversion of PHASE 0. PHASE 1 is high when PHASE 0 is low and vice versa.

PHASE 1 is inverted and gated by DMA′ high to become the 1 MHz PHASE 0 clockpulse input to the 6502. As such, its frequency determines the execution time of instructions in the Apple computer. The duration of a PHASE 0 or PHASE 1 cycle is equal to the duration of a 6502 cycle. This duration is .98 microseconds in a normal cycle and 1.12 microseconds in a long cycle.

The PHASE 0 cycle period is almost coincident with a 6502 machine cycle but slightly leads it. Speaking of PHASE 1 and PHASE 0 as positive gating signals, PHASE 1 occurs approximately during the first half of the 6502 machine cycle and PHASE 0 occurs approximately during the second half. The time relationships of PHASE 1, PHASE 0, and the 6502 machine cycle are shown in Figure 3.5.

Clockpulse action takes place when the 6502 PHASE 0 clockpulse input line switches from high to low or low to high. These transitions trigger actions inside the 6502 which will be discussed in greater detail in the next chapter. A high to low transition of PHASE 0 causes the 6502 to begin a new machine cycle after a short delay.

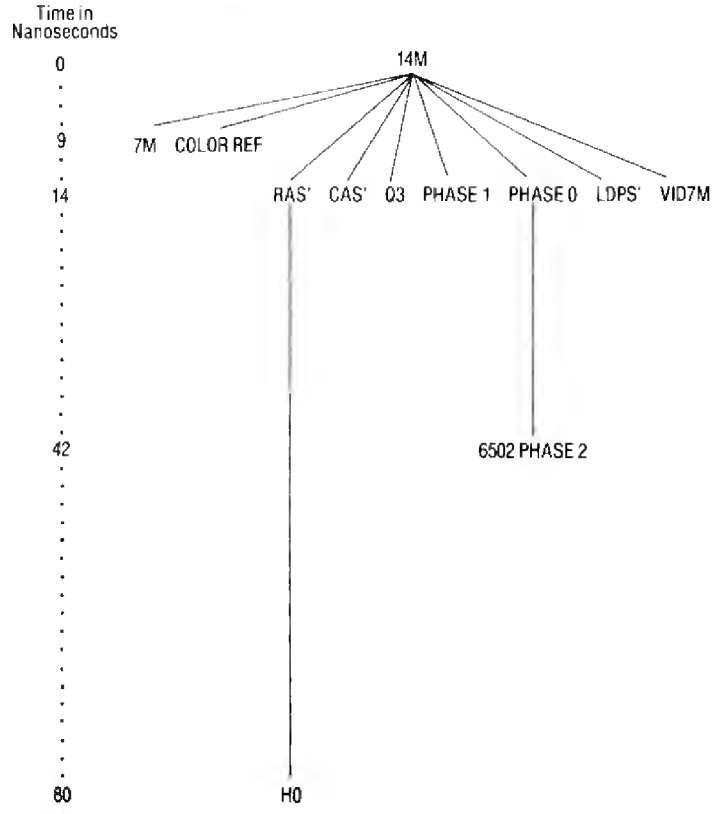


Figure 3.3 Propagation Delay Hierarchy.

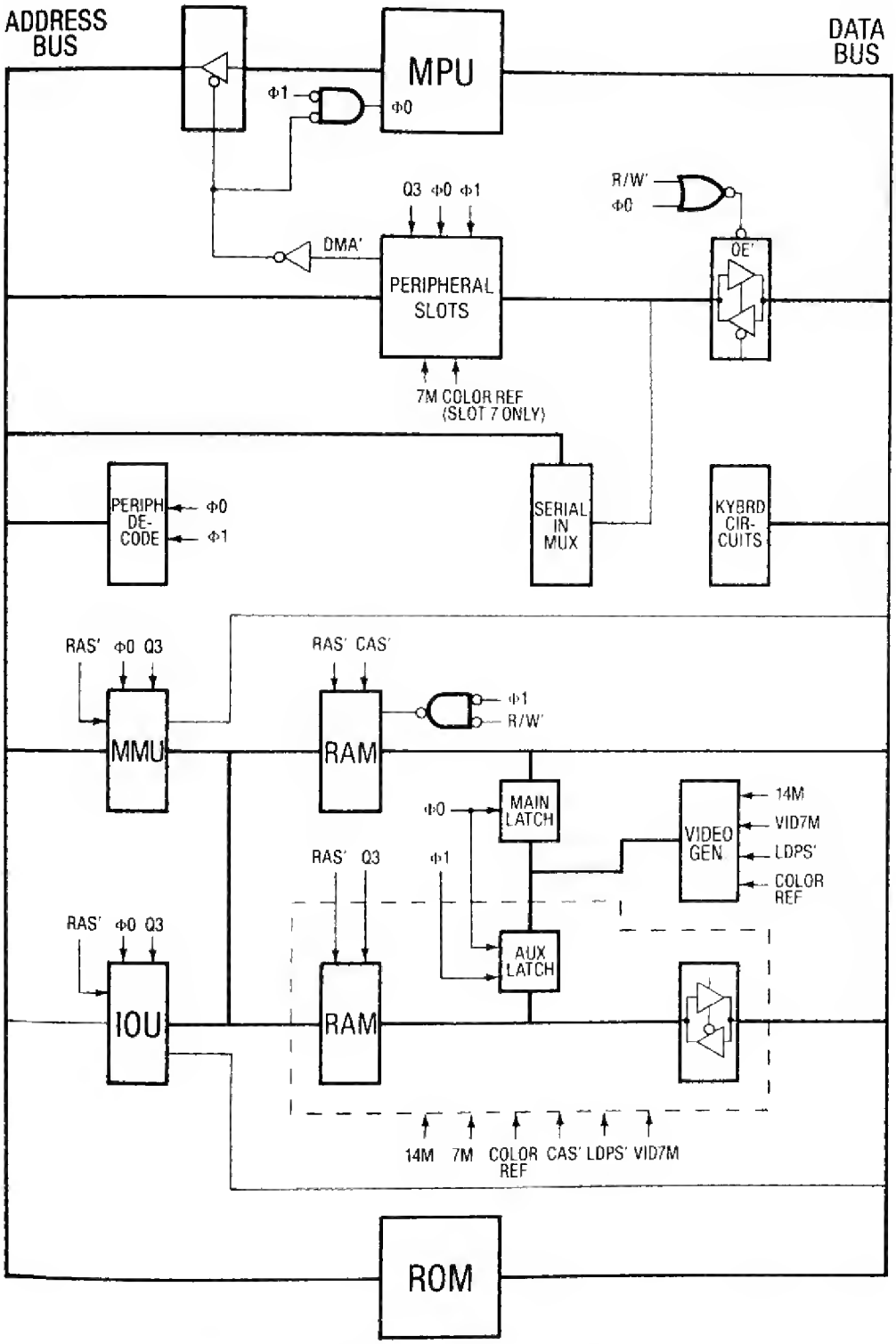


Figure 3.4 Distribution of Timing Generator Outputs.

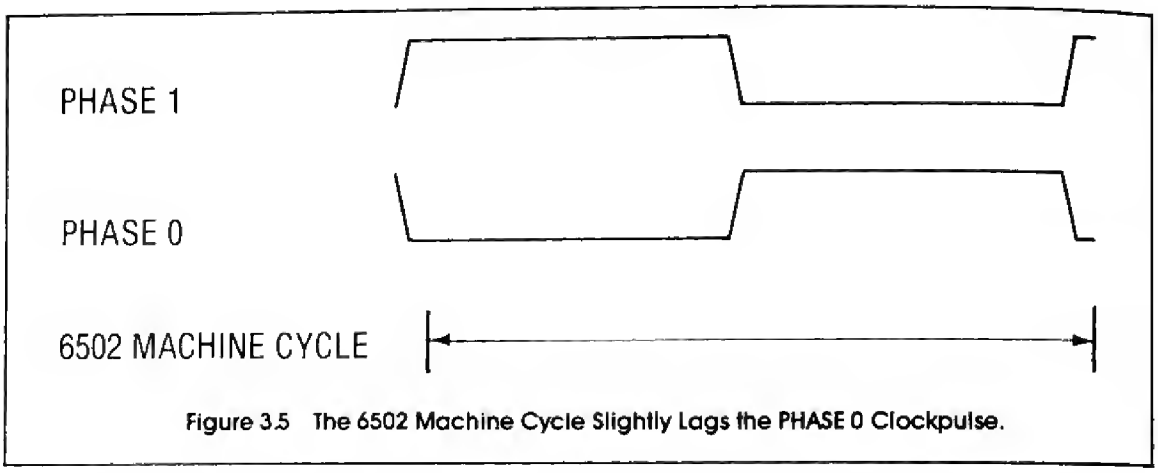


Figure 3.5 The 6502 Machine Cycle Slightly Lags the PHASE 0 Clockpulse.

In addition to triggering 6502 events, PHASE 0 and PHASE 1 are used as a time reference on the motherboard. During PHASE 0, the 6502 address is valid so address decoding from the address bus takes place during PHASE 0. RAM is addressed by the MPU during PHASE 0, and by the video scanner during PHASE 1. Video data from RAM is latched at PHASE 0 rising, and the video data bus contains latched RAM data from the auxiliary card during PHASE 0, and from the motherboard during PHASE 1. Also, since scanner access is during PHASE 1, the RAM read/write control is set to "read" during PHASE 1, even if the 6502 R/W' line is set to "write."

14M, 7M, and COLOR REFERENCE

14M and COLOR REFERENCE (3.5M) are utility clocks which are used in the generation of video. The frequency of Apple video can be as high as 7 MHz, so generating the video signal requires fast clocks. 7M is a utility clock available at the peripheral slots, but not used on the motherboard except in the timing HAL.

14M, 7M, and COLOR REFERENCE are unaffected by the long cycle and have fixed frequencies of 14.318180 MHz, 7.15909 MHz and 3.579545 MHz respectively. 14M is used strictly as a clockpulse in the **video generator**, but COLOR REFERENCE is used differently. Short bursts of the COLOR REFERENCE signal are placed on the video output line once every horizontal scan. A television set is capable of reproducing the continuous COLOR REFERENCE signal from these short bursts, allowing the COLOR REFERENCE input to the television to become the phase reference for color generation. The Apple produces color on a television by shifting the PICTURE signal in relation to the COLOR REFERENCE.*

7M is available at pin 36 of the peripheral slots. COLOR REFERENCE is available at pin 35 of Slot 7 only. 14M is not available at the peripheral slots.

RAS', CAS', and Q3

RAS', CAS', and Q3 are 2 MHz signals. Q3 is used as a timing reference in the MMU and IOU, and is available at the peripheral slots. It is named Q3 because it is identical to the Q3 signal (the Q3 output of a 74S195) of the Apple II. Q3 is also used to strobe the COLUMN address to auxiliary card RAM.

RAS' and CAS' are RAM timing signals that strobe the ROW and COLUMN addresses to motherboard RAM. It can be seen from Figure 3.2 that a RAS'/CAS' sequence occurs twice every 6502 cycle. The PHASE 1 sequence controls the video scanner access to RAM, and the PHASE 0 sequence controls the MPU access to RAM. The falling edges of RAS' and CAS' strobe the ROW address and COLUMN address to RAM, while RAS' selects ROW or COLUMN address lines at the multiplexed address outputs of the IOU and MMU. There is a continuing cycle of RAM access:

1. Select ROW address via RAS' high.
2. Strobe ROW address via RAS' falling.
3. Select COLUMN address via RAS' low.
4. Strobe COLUMN address via CAS' falling.

RAS' is wired directly to all of the motherboard and auxiliary card RAM chips, and to the IOU and

*This book refers to the signal which controls the intensity of the Apple display as the PICTURE signal. When the PICTURE signal is at the white level, the electron beam in the television picture tube strikes the picture screen with enough intensity to cause light emission. The PICTURE signal, SYNC, and COLOR BURST are the three components of the Apple VIDEO signal. More information on this subject is contained in Chapter 8.

MMU as well. It serves as a general timing reference in the IOU and MMU, and RAS' rising during PHASE 1 is the clockpulse which increments the video scanner.

CAS' is connected directly to the eight motherboard RAM chips. It always falls once during PHASE 1, but it only falls during PHASE 0 if CASEN' from the MMU is low. When the MPU is accessing motherboard RAM, the MMU holds CASEN' low, enabling CAS' during PHASE 0 and subsequent data transfer between motherboard RAM and the data bus. When the MPU is accessing any device other than motherboard RAM, the MMU holds CASEN' high, disabling CAS' during PHASE 0, and isolating motherboard RAM from the data bus.

CAS' is not used as the COLUMN address strobe or the RAM enabling signal on the auxiliary card RAM. Q3 is the auxiliary card COLUMN address strobe, and communication between the data bus and auxiliary card RAM is enabled or disabled at the auxiliary RAM card bidirectional data bus driver. The enable/disable signal for this function is EN80' from the MMU.

The three signals which provide the timing reference in the the custom ICs are PHASE 0, RAS', and Q3. The relationships of these signals and some major events that they control are illustrated in Figure 3.6. Remember that in all instances, the events will occur substantially later than their gating inputs because of the long propagation delays in the MMU and IOU.

LDPS' and VID7M

LDPS' and VID7M are timing signals used in the generation of video. These signals vary considerably with the Apple display mode, and they are discussed in greater detail in Chapter 8 than they are here.

The generation of the PICTURE signal is a load/shift process. Text or graphics patterns are loaded from a ROM which is addressed by latched RAM data. The patterns are then shifted out as the PICTURE signal. LDPS' is the load/shift reference for PICTURE signal generation. While LDPS' is low, patterns are loaded in the video generator. While LDPS' is high, they are shifted out. LDPS' always drops low near the end of PHASE 1. In DOUBLE-RES video modes, LDPS' also drops low near the end of PHASE 0.

VID7M is the clockpulse enable signal for the PICTURE signal load/shift register. When VID7M is low, 14M rising causes the register to load or shift. In TEXT40 and HIRES40 display modes, VID7M is a 7 MHz signal (thus the name VID7M). This 7 MHz signal enables loading or shifting every other 14M rising, and is usually identical to the 7M clock, but in

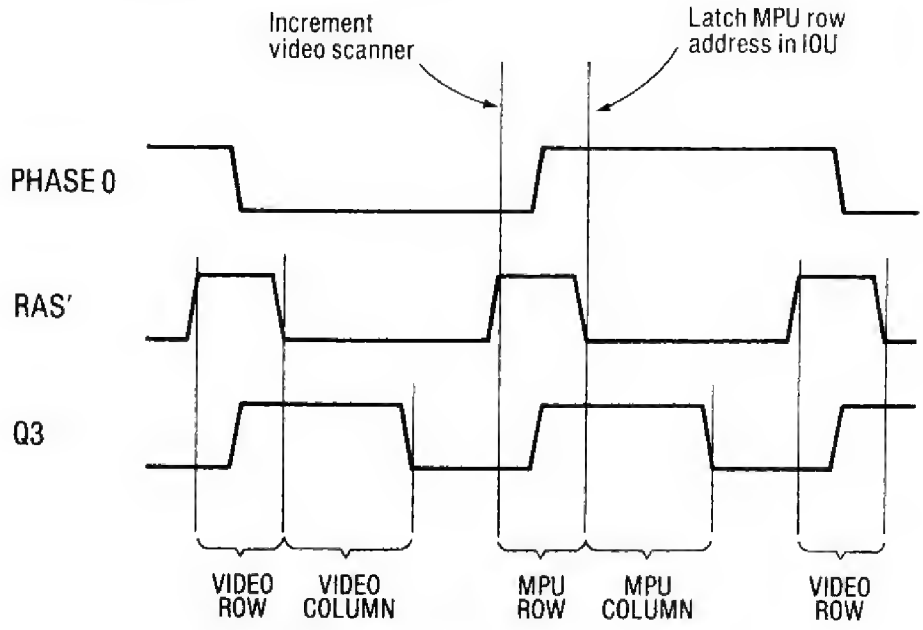


Figure 3.6 Timing Diagram: MMU and IOU Signals.

HIRES40 delayed video cycles, VID7M is the inversion of the 7M clock.

In the LORES40 and DOUBLE-RES display modes, VID7M is a constant low. This enables pattern loading or shifting every time 14M rises, so patterns are shifted out twice as fast as in TEXT40 and HIRES40 display modes.

TELEVISION SCANNING

To understand the operation of the video scanner, it is necessary to understand a little bit about television operation.* The television display is achieved by scanning an electron beam across the screen. The PICTURE signal level controls the beam intensity and the resulting light intensity as the viewer sees it.

The electron beam scans much faster horizontally than it does vertically, so the scan or **raster** is made up of many nearly horizontal lines as shown in Figure 3.7. The scanning circuitry is internal to the television, but the signal input synchronizes the scanning with horizontal and vertical sync. The horizontal sync causes the beam to return very quickly to the left side of the screen, and the vertical sync causes the beam to return very quickly to the top of the screen. The horizontal and vertical sync must occur approximately at television horizontal and vertical frequencies for the television to become synced. In American television, the horizontal scanning frequency is 15,734 Hz and the vertical scanning frequency is 59.94 Hz.

Horizontal and vertical sync occur while the PICTURE signal is at a black, or **blanking**, level. After the horizontal sync causes the beam to go to the left side, the beam traces left to right while the PICTURE signal controls beam intensity.

The Apple must generate the television signal which is a combination of horizontal sync, vertical sync, picture level, and a color burst. It does this by scanning memory for video output with a counter which has recurring periods approximately equal to the horizontal and vertical periods of a television. This counter is the video scanner.

*Chapter 8 contains a more detailed description of television operation. The important concepts here are television scanning and synchronization.

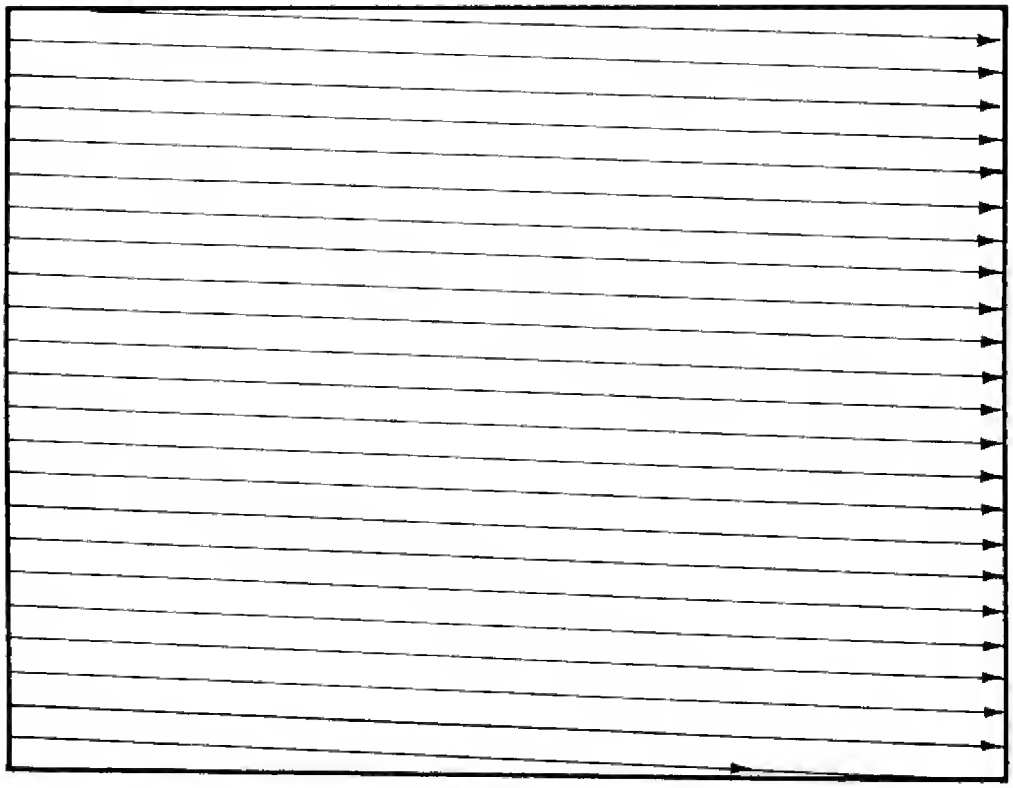


Figure 3.7 Exaggerated View of a Television Scan.

THE VIDEO SCANNER

The video scanner is a counter inside the IOU that counts like a television scans (see Figure 3.8). The low order bits (HPE'-H5-H4-H3-H2-H1-H0) form the horizontal section which sequences through its counts once for every horizontal scan. The high order bits (V5-V4-V3-V2-V1-V0-VC-VB-VA) form the vertical section which sequences through its counts once every vertical scan. In the IOU, outputs of the video scanner are used to develop horizontal and vertical sync for the video signal.

Since states of the video scanner synchronize the television scan, the video scanner can be thought of as scanning the TV screen as it scans memory. The electron beam is always in the same spot on the screen when a given memory location is accessed by the scanner.

The video scanner increments when RAS' rises during PHASE 1. Just like the MPU, the scanner operates at 1 MHz. There is a 1-microsecond period for which every state of the scanner is held until the scanner increments to the next state. During one microsecond, the electron beam travels the width of one TEXT40 character, one LORES block, or seven HIRES40 dots.

Table 3.2 shows the states of the horizontal and vertical sections as well as some events that are initiated at certain states. The vertical states are shown in groups of four because of limited space. The events include sync, the color burst, MIXED mode switching between GRAPHICS and TEXT, VBL (Vertical BLanking), and HBL (Horizontal BLanking). The purpose of this table is to present an overview of the video scanner as it controls events related to the display scan. The nature of these events is discussed in Chapter 8. The details of memory scanning are discussed in Chapter 5.

Horizontal Scanning

The video scanner is divided into the horizontal section and the vertical section. The horizontal section is made up of H0—H5 plus HPE' (Horizontal Preset Enable). These seven bits are mechanized as a 65-state counter which increments every other time RAS' rises. The 65 states of the horizontal counter are 0000000 and 1000000 through 1111111. HPE' is low only during one of the 65 states (0000000), and when it goes low, it causes the horizontal section to preset to 1000000.

One horizontal SYNC pulse is output from the IOU for every time the horizontal section of the video scanner goes through its 65-state sequence, so the 65-state sequence represents one horizontal scan. During 40 of the states, picture information is output on the video line. During the remaining 25 states, the picture is blanked. The blanking period includes the left margin, right margin, and retrace (quick movement of the beam from right to left).

The duration of the horizontal sequence is equal to 64 normal 6502 cycles and one long cycle. This takes 63.695 microseconds, which gives a horizontal frequency of 15,700 Hz. This is very close to the standard television horizontal frequency of 15,734 Hz.

H0, the least significant bit of the video scanner, is output directly to pin 40 of the IOU. No other scanner bits are IOU outputs, but some of them are delayed and output as SEGA, SEGB, and SEGC. Also, outputs such as the multiplexed RAM address, GR+2, WNDW' and others are gated by the scanner and reflect the scanner states.

Vertical Scanning

The vertical section of the video scanner is made up of VA—VC and V0—V5. The vertical section increments every time there is an overflow from the horizontal section, meaning it increments when the horizontal count is 1111111 just before HPE' goes low. The vertical section counts horizontal scans.

The nine bits of the vertical section are mechanized as a 262-state counter. The 262 states are 011111010—111111111. It is a straightforward binary counter which presets on overflow to 011111010. A typical vertical count sequence is

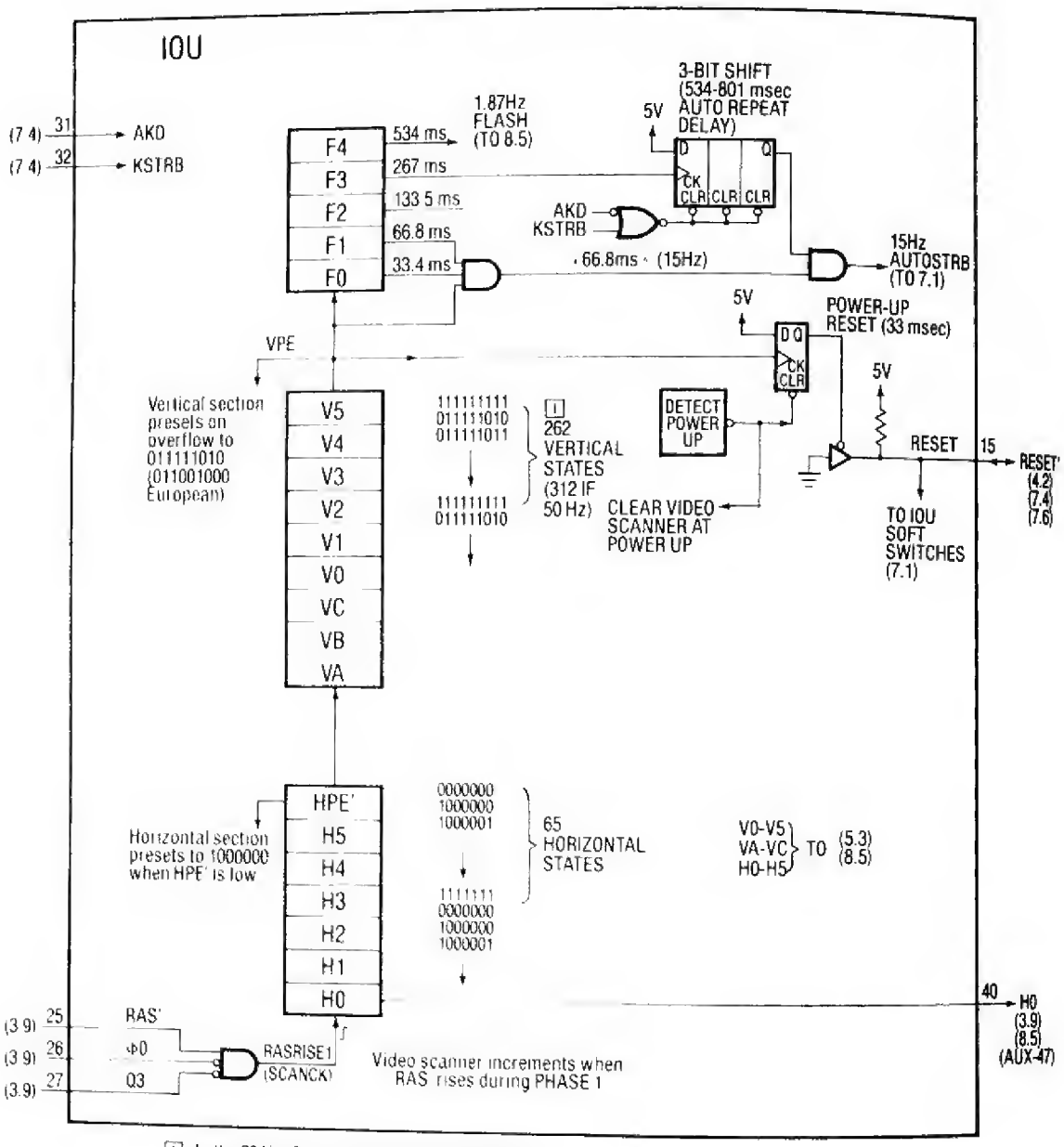
VERTICAL	HORIZONTAL
111100000	1111111
111100001	0000000
111100001	1000000

The vertical preset sequence is

VERTICAL	HORIZONTAL
111111111	1111111
011111010	0000000
011111010	1000000

Once each vertical sequence, the IOU sends vertical sync, so the 262-state sequence represents a vertical scan. During 192 of the scanner states, picture information is output on the video line. The 70 blanked horizontal lines represent the top margin, the bottom margin, and the retrace to the top of the screen.

There are exactly 17030 (65 x 262) 6502 cycles in every television scan of an American Apple. The duration of the television scan is equal to 262 horizontal scans. This is 16,688 microseconds which



1 In the 50 Hz IOU, the vertical section presets to 011001000, resulting in 312 vertical states

Figure 3.8 Functional Diagram: The Video Scanner.

Table 3.2 Video Scanner States (1 of 2).

HORIZONTAL SCANNING			AMERICAN VERTICAL SCANNING			EUROPEAN VERTICAL SCANNING		
HOR. SECTION P 543 210	CK NO	HOR. EVENT	VERTICAL SECTION 543 210 CBA	DISPLAY LINE NO	VERTICAL EVENTS	VERTICAL SECTION 543 210 CBA	DISPLAY LINE NO	VERTICAL EVENTS
1 001 100	53	BURST						
1 001 101	54	BURST						
1 001 110	55	BURST						
1 001 111	56	BURST	111 100 1XX	228-231		011 010 1XX	268-271	
1 010 000	57		111 101 0XX	232-235		011 011 0XX	272-275	
1 010 001	58		111 101 1XX	236-239		011 011 1XX	276-279	
1 010 010	59		111 110 0XX	240-243		011 100 0XX	280-283	TEXT
1 010 011	60		111 110 1XX	244-247		011 100 1XX	284-287	↓
1 010 100	61		111 111 0XX	248-251		011 101 0XX	288-291	
1 010 101	62		111 111 1XX	252-255	PRESET	011 101 1XX	292-295	
1 010 110	63		011 111 01X	256-257		011 110 0XX	296-299	
1 010 111	64		011 111 1XX	258-261		011 110 1XX	300-303	
1 011 000	00	HBL'	100 000 0XX	000-003	VBL', GR	011 111 0XX	304-307	
1 011 001	01	↓	100 000 1XX	004-007	↓ ↓	011 111 1XX	308-311	
1 011 010	02		100 001 0XX	008-011		100 000 0XX	000-003	VBL', GR
1 011 011	03		100 001 1XX	012-015		100 000 1XX	004-007	↓ ↓
1 011 100	04		100 010 0XX	016-019		100 001 0XX	008-011	
1 011 101	05		100 010 1XX	020-023		100 001 1XX	012-015	
1 011 110	06		100 011 0XX	024-027		100 010 0XX	016-019	
1 011 111	07		100 011 1XX	028-031		100 010 1XX	020-023	
1 100 000	08		100 100 0XX	032-035		100 011 0XX	024-027	
1 100 001	09		100 100 1XX	036-039		100 011 1XX	028-031	
1 100 010	10		100 101 0XX	040-043		100 100 0XX	032-035	
1 100 011	11		100 101 1XX	044-047		100 100 1XX	036-039	
1 100 100	12		100 110 0XX	048-051		100 101 0XX	040-043	
1 100 101	13		100 110 1XX	052-055		100 101 1XX	044-047	
1 100 110	14		100 111 0XX	056-059		100 110 0XX	048-051	
1 100 111	15		100 111 1XX	060-063		100 110 1XX	052-055	
1 101 000	16		101 000 0XX	064-067		100 111 0XX	056-059	
1 101 001	17		101 000 1XX	068-071		100 111 1XX	060-063	
1 101 010	18		101 001 0XX	072-075		101 000 0XX	064-067	
1 101 011	19		101 001 1XX	076-079		101 000 1XX	068-071	
1 101 100	20		101 010 0XX	080-083		101 001 0XX	072-075	
1 101 101	21		101 010 1XX	084-087		101 001 1XX	076-079	
1 101 110	22		101 011 0XX	088-091		101 010 0XX	080-083	
1 101 111	23		101 011 1XX	092-095		101 010 1XX	084-087	
1 110 000	24		101 100 0XX	096-099		101 011 0XX	088-091	
1 110 001	25		101 100 1XX	100-103		101 011 1XX	092-095	
1 110 010	26		101 101 0XX	104-107		101 100 0XX	096-099	
1 110 011	27		101 101 1XX	108-111		101 100 1XX	100-103	
1 110 100	28		101 110 0XX	112-115		101 101 0XX	104-107	
1 110 101	29		101 110 1XX	116-119		101 101 1XX	108-111	
1 110 110	30		101 111 0XX	120-123		101 110 0XX	112-115	
1 110 111	31		101 111 1XX	124-127		101 110 1XX	116-119	
						101 111 0XX	120-123	
						101 111 1XX	124-127	

Table 3.2 Video Scanner States (2 of 2).

HORIZONTAL SCANNING		
HOR. SECTION P 543 210	CK NO	HOR. EVENT
1 111 000	32	
1 111 001	33	
1 111 010	34	
1 111 011	35	
1 111 100	36	
1 111 101	37	
1 111 110	38	
1 111 111	39	VERT+1
0 000 000	40	HBL
1 000 000	41	
1 000 001	42	
1 000 010	43	
1 000 011	44	
1 000 100	45	
1 000 101	46	
1 000 110	47	
1 000 111	48	
1 001 000	49	SYNC
1 001 001	50	SYNC
1 001 010	51	SYNC
1 001 011	52	SYNC

AMERICAN VERTICAL SCANNING		
VERTICAL SECTION 543 210 CBA	DISPLAY LINE NO	VERTICAL EVENTS
110 000 0XX	128-131	
110 000 1XX	132-135	
110 001 0XX	136-139	
110 001 1XX	140-143	
110 010 0XX	144-147	
110 010 1XX	148-151	
110 011 0XX	152-155	
110 011 1XX	156-159	
110 100 0XX	160-163	TEXT
110 100 1XX	164-167	
110 101 0XX	168-171	
110 101 1XX	172-175	
110 110 0XX	176-179	
110 110 1XX	180-183	
110 111 0XX	184-187	
110 111 1XX	188-191	
111 000 0XX	192-195	VBL, GR
111 000 1XX	196-199	
111 001 0XX	200-203	
111 001 1XX	204-207	
111 010 0XX	208-211	
111 010 1XX	212-215	
111 011 0XX	216-219	
111 011 1XX	220-223	
111 100 0XX	224-227	SYNC, TXT

EUROPEAN VERTICAL SCANNING		
VERTICAL SECTION 543 210 CBA	DISPLAY LINE NO	VERTICAL EVENTS
110 000 0XX	128-131	
110 000 1XX	132-135	
110 001 0XX	136-139	
110 001 1XX	140-143	
110 010 0XX	144-147	
110 010 1XX	148-151	
110 011 0XX	152-155	
110 011 1XX	156-159	
110 100 0XX	160-163	TEXT
110 100 1XX	164-167	
110 101 0XX	168-171	
110 101 1XX	172-175	
110 110 0XX	176-179	
110 110 1XX	180-183	
110 111 0XX	184-187	
110 111 1XX	188-191	
111 000 0XX	192-195	VBL, GR
111 000 1XX	196-199	
111 001 0XX	200-203	
111 001 1XX	204-207	
111 010 0XX	208-211	
111 010 1XX	212-215	
111 011 0XX	216-219	
111 011 1XX	220-223	
111 100 0XX	224-227	TEXT
111 100 1XX	228-231	
111 101 0XX	232-235	
111 101 1XX	236-239	
111 110 0XX	240-243	
111 110 1XX	244-247	
111 111 0XX	248-251	
111 111 1XX	252-255	PRESET
011 001 0XX	256-259	GR
011 001 1XX	260-263	
011 010 0XX	264-267	SYNC

NOTE: Shaded areas indicate display blanking.

gives a vertical frequency of 59.92 Hz. This is very close to the standard American television vertical frequency of 59.94 Hz.

In a standard television picture, alternating vertical scans are interlaced. This means that every other downward scan is displaced vertically half of the distance between two horizontal scans. Interlacing gives an effective vertical resolution of 525 lines. There is no vertical interlace in the Apple display. This accounts for a disparity in vertical/horizontal frequency relationships between Apple video and broadcast television video. In the Apple, the horizontal frequency is 262 times the vertical frequency. In American broadcast television, the horizontal frequency is 262.5 times the vertical frequency.

Export Apples and the Video Scanner

The television systems of many countries, including those of Europe, have **50 Hz** scanning rates instead of the 60 Hz rate of America. The Apple IIe can be made to support 50 Hz television scanning by installing a **14.25 MHz crystal** and a **50 Hz IOU**.

The lower frequency crystal changes the period of a horizontal scan from about 63.7 microseconds to about 64 microseconds. The 50 Hz IOU adds 50 horizontal scans to the vertical scan to yield a vertical rate of about 50 Hz.

The 50 Hz IOU and 14.25 MHz crystals are installed in special motherboards that have PAL (Phase Alternating Lines) color encoding circuitry built-in. PAL is a 50 Hz television system used in many countries including all major western European countries except France. As of this writing, the 14M oscillator is made of discrete circuits and the crystal used is 14.25045 MHz (see Figure 3.9). However, Apple has developed a hybrid oscillator which is used in the Apple IIc and will probably see use in the Apple IIe. The frequency in the export version of the hybrid oscillator is 14.25 MHz. Both of these frequencies yield approximate horizontal scan durations of 64 microseconds (63.998 from 14.25045 and 64.000 from 14.25).

In the 50 Hz IOU, the vertical section of the video scanner presets on overflow to 011001000 instead of 011111010. There are 312 states represented by 011001000—111111111. This gives a vertical frequency of 50.08 Hz. Even though there are 50 extra horizontal scans in the 50 Hz Apple, there is no extra vertical resolution. In either scanning system, there are 192 horizontal scans in which picture information is displayed.

The Flash Counter and Power-up Reset Circuit

F0—F4 of Figure 3.8 make up the flash counter. This counter counts television scans, and I call it the flash counter because F4 is used to switch flashing text between NORMAL and INVERSE. Flashing text doesn't necessarily have to be in sync with the display scan, but the video scanner provides a handy uninterrupted recurring signal (the scanner overflow) which the IOU uses for a time reference. Other functions which depend on the flash counter for a time reference are the delay before activating the keyboard auto repeat function, the frequency of the keyboard auto repeat function, and the time-out period of the power-up reset.

The flash counter is not mentioned in any published Apple literature that I know of. The Figure 3.8 representation and the "flash counter" and "F0—F4" nomenclature are mine, not Apple's. The reason for my assumption of the existence of the flash counter is that the flashing text, power-up reset, and auto repeat functions always toggle just after a video scanner overflow. Also, these features operate at frequencies that suggest they are controlled by a simple binary counter incrementing, or perhaps decrementing, when the video scanner overflows.

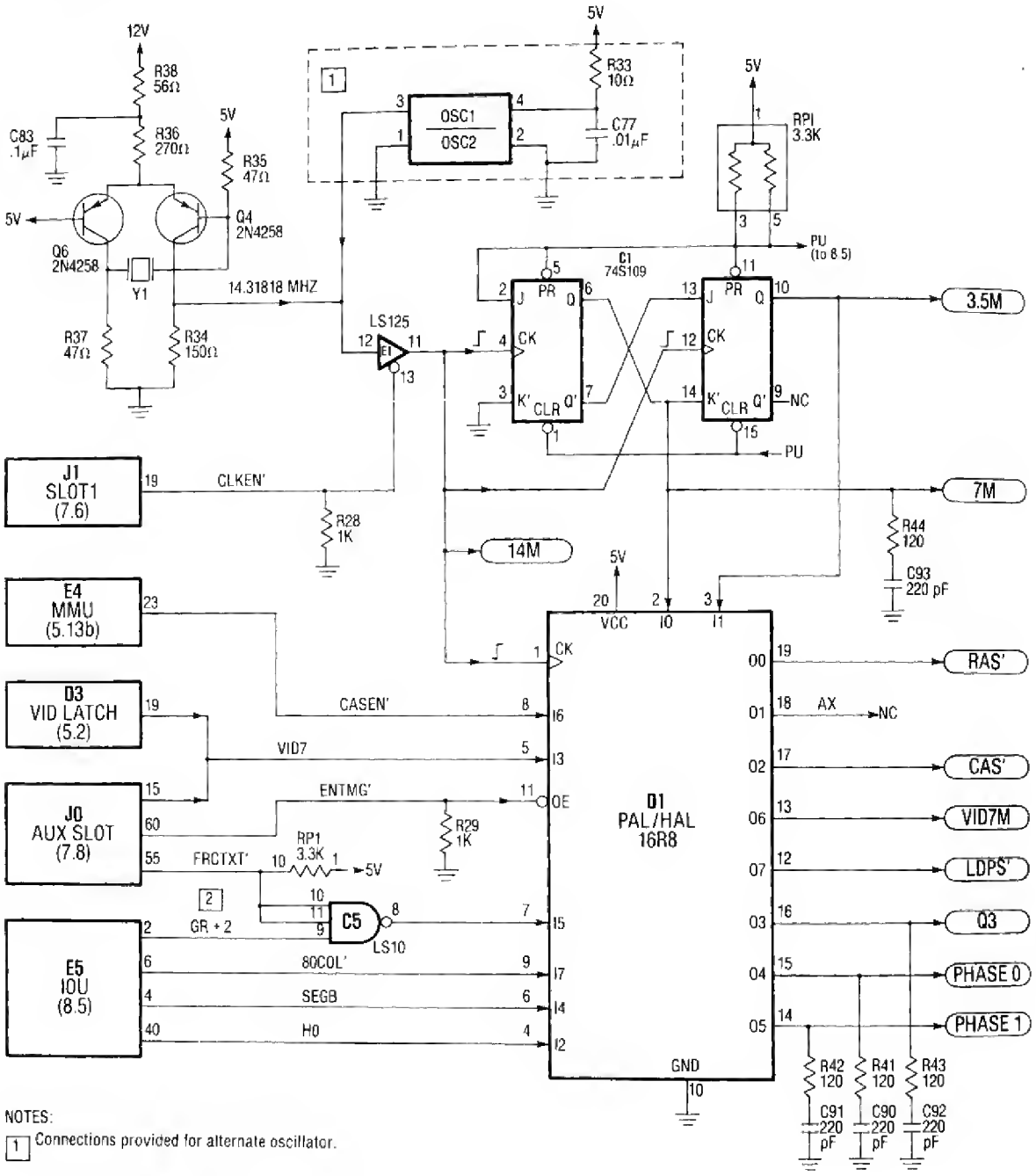
Another circuit not mentioned in Apple literature is the power-up reset circuit. When the Apple IIe is first turned on, the IOU holds the RESET' line low for about 33 milliseconds. If you prevent the video scanner from counting by pulling CLKEN' high, the RESET' line stays low until you enable the 14M clock and let the scanner count for a while. When the RESET' line does go high, it does so approximately when the video scanner overflows, as closely as I can observe. My deduction is that the video scanner presets to 000000000/0000000 at power-up, and that the RESET' line is allowed to rise 32.6 milliseconds later when the scanner overflows for the first time. The Figure 3.8 power-up reset circuit will generate such a reset.*

Figure 3.8 shows generation of an AUTOSTRB signal which is an artificial keyboard strobe. The KEYSTROBE soft switch is set when either KSTRB (the real keyboard strobe) or AUTOSTROBE goes high. When a key is held for 534—801 milliseconds, the AUTOSTRB starts to alternate at 15 Hz, repeatedly setting the KEYSTROBE soft switch (Figure 7.1)to simulate rapid keypresses.

F3 of the flash counter is the clockpulse for generating the delay before auto repeat. This can be deduced from the 267-millisecond variation in the delay. The delay could be produced from a 2-bit counter or from a 3-bit shift register like the one shown in Figure 3.8. In either case, KSTRB must reset the delay generator so that pressing a key interrupts the auto repeat function until the delay times out again.

The variation in duration of the delay before auto repeat is a mild problem for me. This delay should be nearly constant if a typist is to become skillful at performing keyboard auto repeat functions. I feel that a variation of .267 seconds here is too great, and that it prevents me from really making the keyboard "sing." The variation could be reduced greatly if the 3-bit shift register in Figure 3.8 were replaced by a 4-bit VPE counter, identical to the first four bits of the flash counter except that it is cleared when KSTRB is high or AKD is low. The overflow from this counter would be an auto repeat enable signal, delayed from the initial keypress by 534—551 milliseconds.

*The power-up reset duration can only be measured with no Disk II controller in any peripheral slot, because the controller 100 msec power-up reset will mask the IOU 33 msec power-up reset.



NOTES:

1 Connections provided for alternate oscillator.

2 On Revision A motherboards, E5-2 is connected directly to D1-7. The addition of FRCTXT' gating in Revision B makes the DOUBLE-RES graphics modes possible.

Figure 3.9 Schematic: The Timing Generator.

The durations of events controlled by the flash counter are generally exact multiples of the duration of a vertical scan. The exception is the power-up reset time out which lasts 512 horizontal scans (the clear-to-overflow period of the vertical section of the video scanner). Durations and frequencies of events controlled by the flash counter are listed in Table 3.3. With the exception of the power-up reset, durations will be about 1.2 times longer in 50 Hz IOUs.

THE LONG CYCLE

The discussions have alluded to the **long cycle** in a limited way, but we are now in a better position to understand the reasons for it.

Video output begins each horizontal scan when the horizontal count reaches 1011000. For color coherency, the video output needs to begin at the same point in relation to COLOR REFERENCE on every scan. Since there are 3.5 COLOR REFERENCE cycles in a video scanner cycle, the phase of COLOR REFERENCE at the start of a video shift alternates 180 degrees each scanner cycle. Because of the 180 degree phase alternation each cycle, a 7-dot HIRES40 pattern represents different colors when it is stored in an even RAM address than when it is stored in an odd RAM address.

There are 65 video scanner cycles per horizontal screen line. Since this is an odd number, there would be an odd number of 180 degree phase alternations per horizontal line. This would cause the starting phase relationship to alternate every horizontal line. By delaying video shift timing half of a COLOR REFERENCE period once every horizontal line, the same beginning phase relationship occurs every horizontal line. As a side effect, all 1 MHz and 2 MHz signals are elongated once every horizontal line.

TIMING GENERATOR HARDWARE

Timing generation in the Apple consists of making a lot out of a little. By this I mean that the 14M clock is divided and processed to make the slower, more complex signals. Most of the processing is performed in the timing HAL.

14M comes from a crystal controlled 14.31818 MHz oscillator via one fourth of a 74LS125 tri-state driver (see Figure 3.9). 14M is pretty symmetrical, but symmetry is not important since only the rising edge of 14M is used in the Apple.

The tri-state 14M driver is always enabled unless the Apple has a special peripheral card installed in Slot 1. It is possible for a Slot 1 card to isolate the 14M line from the motherboard oscillator by bringing CLKEN′ high. An auxiliary card can then substitute its own master clock signal for the disabled motherboard 14M signal. With peripheral Slot 1 empty or a peripheral card with pin 19 open installed in Slot 1, the CLKEN′ line is open and pull-down resistor R28 keeps the 14M tri-state driver enabled.

The CLKEN′ feature could be used by diagnostic cards designed to check out and troubleshoot Apples. It could also be used in some mad hacker scheme too insane for me to envision. If a mad hacker happens to read this, my advice, if you want to change the 14M frequency, is to ignore the CLKEN′ line and change the crystal.

7M and COLOR REFERENCE generation is straightforward frequency division. 7M is 14M divided by two. COLOR REFERENCE is 7M divided by two. The connections are such that COLOR REFERENCE toggles when 7M falls (see Figure 3.2). The frequency division takes place in a 74S109 dual flip-flop. An S109 is used here instead of an LS109 because the S109 has more driving power, and 7M is distributed to all of the peripheral slots.

Table 3.3 Events Controlled by the Flash Counter.

EVENT	DURATION/ FREQUENCY	REMARKS
Power-up reset	32.6 msec	512 horizontal scans
Flash cycle	1.87 Hz	Vertical freq./32
Delay before auto repeat	534—801 msec	32-48 vertical scans
Auto repeat frequency	15 Hz	Vertical freq./4

14M, 7M, and COLOR REFERENCE are inputs to the HAL. This single IC generates all of the remaining timing signals—PHASE 0, PHASE 1, RAS', CAS', Q3, LDPS', and VID7M.

The Timing HAL

HAL (Hard Array Logic) and PAL (Programmable Array Logic) are a relatively recent development in microelectronics. They are skeletal logic structures whose actual logic functions can be specified, in the case of HAL, or programmed, in the case of PAL.

The Monolithic Memories series of HAL and PAL is available in a variety of skeletal structures having STTL signal I/O characteristics. An engineer can choose a HAL/PAL, then design and debug his application using field programmable PALs. If there is enough volume to merit it, the debugged IC can then be ordered directly from the manufacturer as HAL. In the case of the Apple IIe, there is, of course, enough volume to merit purchase of HAL from the manufacturer.

The HAL used in the Apple IIe is a HAL16R8 which contains eight D flip-flops fed by multiple and/or input logic arrays. The flip-flops are all clocked by 14M rising, so the HAL outputs all experience approximately the same propagation delay from 14M rising (about 14 nanoseconds). The 16R8 outputs are tri-state, and the outputs are disabled if an auxiliary card brings the pulled down ENTMG' line high. All timing generator signals are connected to the auxiliary slot so an auxiliary card can substitute its own signals for the PAL outputs. The ENTMG' line is open on every auxiliary slot card that I know of.

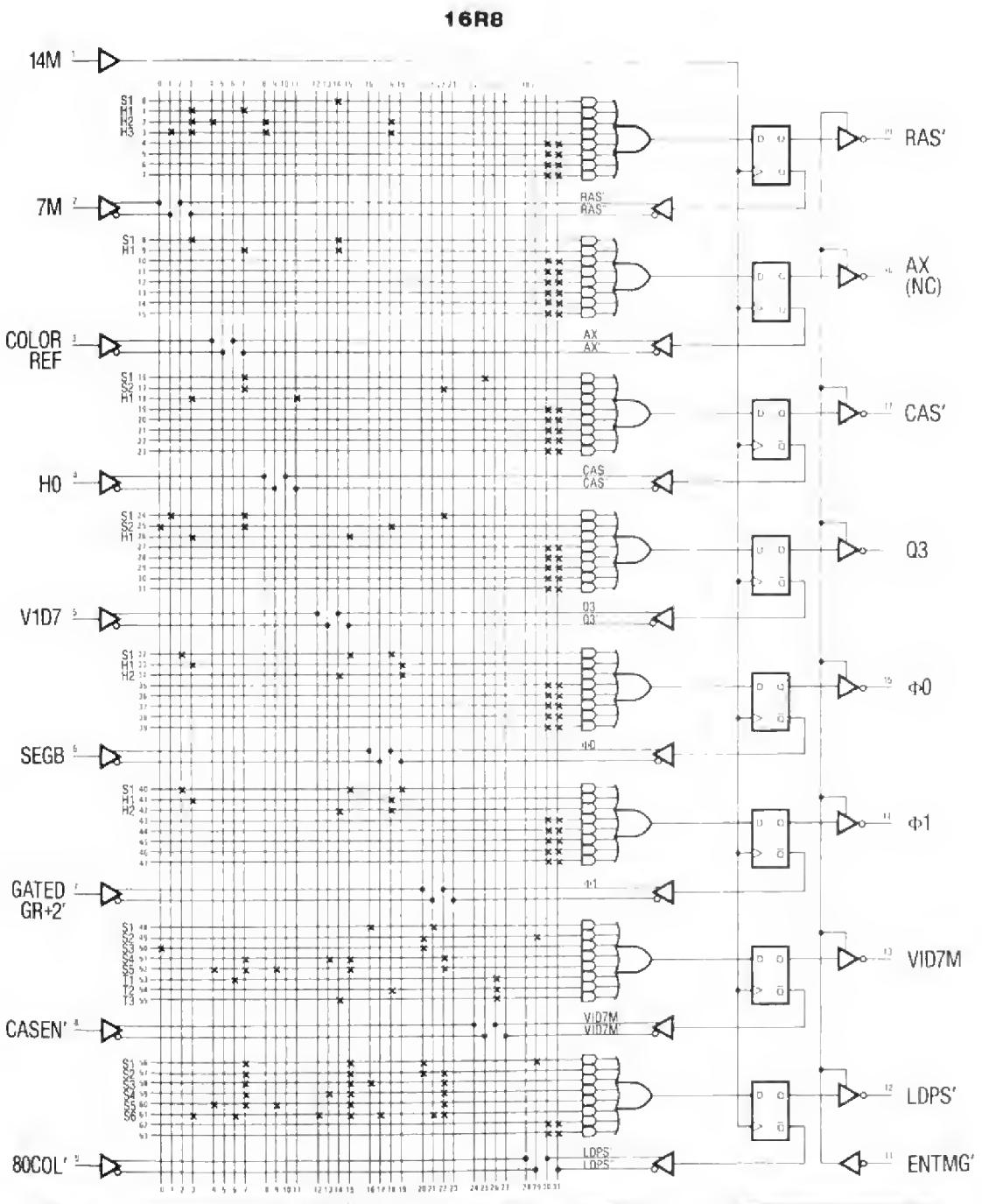
Figure 3.10 is a 16R8, programmed to operate as the Apple IIe timing HAL. I filled in the X's to match the timing signal characteristics of the Revision B Apple IIe. The Revision A HAL must be different than the Revision B HAL, because GR+2 is distributed to the Rev A HAL and gated GR+2' is distributed to the Rev B HAL. The GR/GR' X's must therefore be swapped in the Rev A HAL layout as opposed to the Rev B HAL layout.*

*A second difference with the Rev A HAL is that the COLOR REFERENCE has a different phase relationship with the other signals than that illustrated in Figure 3.2. The Rev A PAL would thus result in non-standard colors if plugged into a Rev B motherboard.

Figure 3.10 is certain to be different than the actual Apple HAL in minor details. There is little room for variation, however, in the substance of the resulting logic equations. It was not at all obvious to me how some of the required logic functions could be performed with the available inputs. I was only able to come up with a working layout after considerable head scratching. Apple's effort in visualizing the timing generator as a HAL application and in producing such an efficient design is impressive.

Table 3.4 is a list of logic equations reduced from Figure 3.10. Most readers will find these equations easier to analyze. Both Table 3.4 and Figure 3.10 are presented here for reference, however, and complete analysis will not be particularly valuable for most readers. A good grasp of the Figure 3.2 timing diagram is much more important in understanding the Apple. For those who are interested in the HAL layout, here are some interesting features.

1. All outputs are the inversion of the flip-flop outputs, so setting a flip-flop causes its output line to go low and vice versa.
2. The RAS', AX, CAS', Q3, PHASE 0, and PHASE 1 logic is best thought of as set/hold logic. The set terms do not have feedback from the flip-flop they are controlling, but the **hold** terms only come true if the flip-flop they are controlling is already set. The flip-flop sets if any set term comes true, and a flip-flop, once set, will stay set if any set term or hold term is true.
3. RAS', AX, CAS', and Q3 are set up as a shift register. If Q3 is high, a low level is shifted to RAS' then AX then CAS' then Q3 (with special logic on Q3 since CAS' won't fall during PHASE 0 if CASEN' is high). If Q3 is low, a high level is shifted to AX then RAS' then simultaneously to CAS' and Q3.
4. The delay logic that causes the long cycle is the H2 and H3 terms of the RAS' flip-flop.
5. CASEN' from the MMU is not PHASE 0 gated, so the CAS' flip-flop must set during PHASE 1 whether CASEN' is high or low (see the S2 term on the CAS' flip-flop).
6. The SEGB, gated GR+2', VID7, and 80COL' inputs to the PAL are used only in generation of LDPS' and VID7M. Also, none of the other HAL outputs are affected by LDPS' and VID7M. LDPS' and VID7M generation is discussed in Chapter 8.



NOTE: Base drawing is from "Bipolar LSI 1984 Databook," fifth edition, reprinted with permission from Monolithic Memories, Inc. The Xs were filled in by Jim Sather.

Figure 3.10 The Timing HAL Layout.

Table 3.4 Timing HAL Logic Equations.

PIN ASSIGNMENTS

ARRAY INPUTS	OUTPUTS	OTHER INPUTS
I0-2 = 7M	Q0′-19 = RAS′	CP-1 = 14M
I1-3 = CLR REF	Q1′-18 = AX	OE′-11 = ENTMG′
I2-4 = H0	Q2′-17 = CAS′	VCC-20 = +5V
I3-5 = VID7	Q3′-16 = Q3 ′	GND-10 = GROUND
I4-6 = SEGB	Q4′-15 = Φ0	
I5-7 = GATED GR+2′	Q5′-14 = Φ1	
I6-8 = CASEN′	Q6′-13 = VID7M	
I7-9 = 80COL′	Q7′-12 = LDPS′	

SIGNAL	EQUATIONS	NOTES
RAS′	S1 = Q3 H1= RAS″ • AX′ H2= RAS″ • CLR REF • H0 • Φ0 H3= RAS″ • 7M′ • H0 • Φ0	FALL AFTER Q3 RISES RISE AFTER AX RISES LONG CYCLE DELAY LONG CYCLE DELAY
AX	S1 = RAS″ • Q3 H1= AX′ • Q3	FALL AFTER RAS′ FALLS RISE AFTER Q3 FALLS
CAS′	S1 = AX′ • CASEN″ S2 = AX′ • Φ1 H1= CAS″ • RAS″	MMU, MAY I? NUTS TO MMU DURING Φ1 RISE AFTER RAS′ RISES
Q3	S1 = AX′ • Φ1 • 7M′ S2 = AX′ • Φ0 • 7M H1= Q3′ • RAS″	AX′ • Φ1 • CAS′ ALSO WORKS CAS′ NO WORKEE RISE AFTER RAS′ RISES
Φ0	S1 = Φ0 • RAS′ • Q3′ H1= Φ0′ • RAS″ H2= Φ0′ • Q3	TOGGLE AT RAS′ • Q3
Φ1	S1 = Φ0′ • RAS′ • Q3′ H1= Φ0 • RAS″ H2= Φ0 • Q3	Φ0 INVERTED
VID7M	S1 = GR″ • SEGB S2 = GR′ • 80COL″ S3 = GR′ • 7M S4 = VID7′ • Φ1 • Q3′ • AX′ S5 = H0′ • CLR REF • Φ1 • Q3′ • AX′ T1 = VID7M • AX T2 = VID7M • Φ0 T3 = VID7M • Q3	LORES GRAPHICS IS HIGH SPEED DOUBLE RES IS HIGH SPEED SAME AS 7M IF NOT HIRES FRCTXT″ • 80COL′ ⟶ 7M, UNDELAYED HIRES DELAY CHECK AT Φ1 • Q3′ • AX′ NO DELAY AT RIGHT DISPLAY EDGE TOGGLE THROUGH AX KEEP TOGGLING THROUGH Φ0 KEEP TOGGLING THROUGH Q3
LDPS′	S1 = Q3′ • AX′ • 80COL″ • GR′ S2 = Q3′ • AX′ • Φ1 • GR′ S3 = Q3′ • AX′ • Φ1 • SEGB S4 = Q3′ • AX′ • Φ1 • VID7′ S5 = Q3′ • AX′ • Φ1 • CLR REF • H0′ S6 = Q3′ • AX • RAS″ • Φ1 • VID7 • SEGB′ • GR″	DOUBLE RES CAUSES DOUBLE LDPS′ TEXT MODE LORES NOT DELAYED HIRES RIGHT DISPLAY EDGE CUTOFF HIRES DELAYED LDPS′

SWITCHING SCREEN MODES IN TIMED LOOPS

A horizontal scan in the Apple takes exactly 65 machine cycles of the 6502. A vertical scan takes exactly 17030 machine cycles. This information can be used to switch screen modes in timed loops to give apparent combination screen modes.

For example, the screen can be split so that half of each horizontal line is LORES and the other half of each horizontal line is HIRES by switching between modes in alternating 33- and 32-cycle loops. Similarly, the screen can be split so that half of all the horizontal lines are LORES and the other half are HIRES by switching back and forth every 8515 cycles. The latter can be accomplished using the sample programs listed in Figures 3.11a and 3.11b. The assembly language program of Figure 3.11a, when assembled, is a subroutine that performs the screen splitting. The BASIC program of Figure 3.11b sets up a color display and calls the machine language subroutine.

The example program causes the Apple to be in LORES for 131 TV lines and in HIRES for 131 TV lines. The display is aligned vertically by holding down any key on the keyboard. The result of running this program is the split screen display pictured in Figure 8.14.

In the Apple IIe, it is possible to read the state of VBL' (the inversion of the Vertical BLanking gate) at address $C019. VBL goes high just after the last displayed address is scanned at the bottom right of the Apple screen, and it goes low at the same horizontal point in the last undisplayed horizontal scan at the top of the screen. Either of these points can be located within an accuracy of seven MPU cycles by simply polling VBL'. For example, when the following polling loop falls through, the display scan will be from zero to six cycles past the end of VBL, and from 19 to 25 cycles before the first display memory is scanned.

```
VBLOFF   EQU $C019    MINUS => VBL'
                      PLUS => VBL
POLL1    LDA VBLOFF
         BMI POLL1    FALL THROUGH
                      AT VBL
POLL2    LDA VBLOFF
         BPL POLL2    FALL THROUGH
                      AT BEGIN VBL'
```

Once this point is located, a program can perform a switching action in the blanking period before any horizontal scan line by waiting for 65 cycles per horizontal scan. The following example provides a stable display of HIRES graphics for the first 96 lines and LORES graphics for the second 96 lines.

```
VBLOFF   EQU $C019      MINUS => VBL', PLUS => VBL
HIRES    EQU $C057
LORES    EQU $C056
POLL1    LDA VBLOFF
         BPL POLL1      FALL THROUGH AT VBL'
         LDA HIRES      4 CYCLES
         LDX #6         2 CYCLES
         JSR WAITX1K    6000 CYCLES (SEE FIG 3.11a)
         LDY #23        2 CYCLES
         JSR WAITX10    230 CYCLES
         LDA LORES      4 CYCLES; 6242 CYCLES = 65 x 96 + 2
POLL2    LDA VBLOFF
         BMI POLL2      FALL THROUGH AT VBL
         BPL POLL1
```

```
SOURCE FILE: SPLIT SCREEN
0000:              1 ********************************************************
0000:              2 *                                                      *
0000:              3 *                 HIRES/LORES SPLIT                    *
0000:              4 *                     8515/8515                        *
0000:              5 *                   BY JIM SATHER                      *
0000:              6 *                     2/15/1983                        *
0000:              7 *                                                      *
0000:              8 ********************************************************
C000:              9 KBD      EQU  $C000
C010:             10 KBDSTRB  EQU  $C010
C054:             11 PAGE1    EQU  $C054
C056:             12 LORES    EQU  $C056
0000:             13 *
0000:             14 * THIS PROGRAM TOGGLES THE HIRES/LORES SWITCH
0000:             15 * EVERY 8515 CYCLES.
0000:             16 *
----- NEXT OBJECT FILE NAME IS SPLIT SCREEN.OBJ0
1F00:             17          ORG  $1F00
1F00:AC 54 C0     18 SPLIT    LDY  PAGE1
1F03:A0 27        19 SLEW     LDY  #39        ;SLEW SCREEN IF KEY PRESSED.
1F05:20 27 1F     20          JSR  WAITX10
1F08:AC 10 C0     21          LDY  KBDSTRB
1F0B:AC 00 C0     22 KEYCHK   LDY  KBD
1F0E:30 F3        23          BMI  SLEW
1F10:69 01        24          ADC  #1         ;TOGGLE HIRES/LORES SWITCH
1F12:29 01        25          AND  #$01
1F14:AA           26          TAX
1F15:BC 56 C0     27          LDY  LORES,X
1F18:A2 08        28          LDX  #8
1F1A:20 31 1F     29          JSR  WAITX1K    ;WAIT 8000 CYCLES
1F1D:A0 31        30          LDY  #49
1F1F:20 27 1F     31          JSR  WAITX10    ;WAIT 490 CYCLES
1F22:18           32          CLC
1F23:90 B6        33          BCC  KEYCHK
1F25:             34 *
1F25:             35 * TIMING ROUTINES:
1F25:             36 * WAITX10 WAITS Y-REG TIMES 10 CYCLES.
1F25:             37 * (MINIMUM WAIT 20 CYCLES)
1F25:             38 * WAITX1K WAITS X-REG TIMES 1000 CYCLES.
1F25:             39 *
1F25:D0 01        40 LOOP10   BNE  SKIP
1F27:88           41 WAITX10  DEY             ;WAIT Y-REG TIMES 10
1F28:88           42 SKIP     DEY
1F29:EA           43          NOP
1F2A:D0 F9        44          BNE  LOOP10
1F2C:60           45          RTS
1F2D:48           46 LOOP1K   PHA
1F2E:68           47          PLA
1F2F:EA           48          NOP
1F30:EA           49          NOP
1F31:A0 62        50 WAITX1K  LDY  #98        ;WAIT X-REG TIMES 1000
1F33:20 27 1F     51          JSR  WAITX10
1F36:EA           52          NOP
1F37:CA           53          DEX
1F38:D0 F3        54          BNE  LOOP1K
1F3A:60           55          RTS

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Figure 3.11a Assembler Listing: Timed Execution Screen Splitting.

Locating VBL within seven cycles may not be accurate enough for your application. It would not suffice for screen mode switching at a specific position during display time. VBL can be located precisely by finding the point where VBL switches from off to on within seven cycles, then slewing backwards in 17029-cycle polling loops until VBL is sensed off. The video scanner state will then be at precisely one cycle before VBL (scanner = 010111111/1111111). VBL switching from on to off can be similary located, and any video scanner state can be located by detecting VBL on or VBL off, and then waiting an appropriate number of cycles. For example, the program in Figure 3.12 will result in a LORES graphics display with a 20-character text message in the middle of the screen.

Any of the screen splitting routines of this application note can be called from BASIC programs or programs written in other languages. Many variations of these routines are possible. Any number of unusual Apple displays can be created with a combination of timed loops and polling for VBL.

```
10  REM
11  REM
12  REM  SET UP LORES AND HIRES AND CALL SPLIT SCREEN.
13  REM
14  REM
20  PRINT  CHR$ (4);"BLOAD SPLIT SCREEN.OBJ0"
30  HGR : HOME : VTAB 21: PRINT "1  7  D  2  8  E  B  4  5  A 3 6 C 9 F 8"
40  DIM COLR(39),X(21)
100  FOR A = 0 TO 39: READ COLR(A): COLOR= COLR(A): VLIN 0,39 AT A: NEXT A
200  FOR A = 0 TO 21: READ COLR(A): READ X(A): HCOLOR= COLR(A)
210  HPLOT X(A),0 TO X(A),159: NEXT A
220  FOR A = 8319 TO 16383 STEP 128: POKE A,64: NEXT A
300  CALL 7936
400  REM  LORES DATA
410  DATA  1,0,7,7,0,13,13,0,2,2,0,0,0,0,14,14,0,11,11,0
420  DATA  4,4,0,0,5,0,0,10,0,3,0,6,0,12,0,9,0,15,0,8
500  REM  HIRES DATA
510  DATA  4,0,3,20,4,21,3,41,4,42,7,62,7,83,7,104,3,105,7,125,3,126,7,159,3,161
520  DATA 7,180,3,182,3,206,7,220,3,233,7,247,3,262,3,263,7,279
```

Figure 3.11b BASIC Listing: Call Split Screen.

```
SOURCE FILE: LIL TEXT WINDOW
0000:              1 ***********************************************
0000:              2 *                                             *
0000:              3 *              LITTLE TEXT WINDOW             *
0000:              4 *                                             *
0000:              5 *     DEMONSTRATES PRECISE VBL DETECTION      *
0000:              6 *                                             *
0000:              7 *           Jim Sather --- 8/15/84            *
0000:              8 *                                             *
0000:              9 ***********************************************
002C:             10 H2       EQU  $2C        HLINE RIGHT TERMINUS
0030:             11 COLOR    EQU  $30        LORES COLOR BYTE
C00C:             12 COL40    EQU  $C00C      80COL RESET ADDRESS
C019:             13 VBLOFF   EQU  $C019      MINUS => VBL', PLUS => VBL
C050:             14 GRAFIX   EQU  $C050
C051:             15 TEXT     EQU  $C051
C056:             16 LORES    EQU  $C056
F819:             17 HLINE    EQU  $F819      LORES HLINE SUBROUTINE
0000:             18 *
----- NEXT OBJECT FILE NAME IS LIL TEXT WINDOW.OBJ0
1F00:             19          ORG  $1F00
1F00:8D 0C C0     20          STA  COL40      SINGLE-RES DISPLAY
1F03:AD 56 C0     21          LDA  LORES
1F06:A9 27        22          LDA  #39        FILL SCREEN USING HLINE
1F08:85 2C        23          STA  H2         RIGHT COORDINATE = 39
1F0A:A9 CC        24          LDA  #$CC
1F0C:85 30        25          STA  COLOR      COLOR = HIRES40 GREEN
1F0E:A2 2F        26          LDX  #47        CLEAR LINES 47-0
1F10:A0 00        27 FILL     LDY  #0         LEFT COORDINATE = 0
1F12:8A           28          TXA             ;GET VERT COORDINATE FROM X
1F13:20 19 F8     29          JSR  HLINE
1F16:CA           30          DEX
1F17:10 F7        31          BPL  FILL
1F19:A2 15        32          LDX  #21        INSERT MESSAGE
1F1B:BD 8E 1F     33 MSGLP    LDA  MSG,X
1F1E:9D B1 05     34          STA  $5B1,X     MESSAGE AT LINE 11, POSITION 10
1F21:CA           35          DEX
1F22:10 F7        36          BPL  MSGLP
1F24:AD 19 C0     37 POLL1    LDA  VBLOFF     FIND END OF VBL
1F27:30 FB        38          BMI  POLL1      FALL THROUGH AT VBL
1F29:AD 19 C0     39 POLL2    LDA  VBLOFF
1F2C:10 FB        40          BPL  POLL2      (2) FALL THROUGH AT VBL'
1F2E:             41 *
1F2E:A5 00        42          LDA  $00        (3) NOW SLEW BACK IN 17029 CYCLE LOOPS
1F30:A2 11        43 LP17029  LDX  #17        (2)
1F32:20 84 1F     44          JSR  WAITX1K    (17000)
1F35:20 8D 1F     45          JSR  RTS1       (12)
1F38:A5 00        46          LDA  $00        (3)
1F3A:A5 00        47          LDA  $00        (3)
1F3C:AD 19 C0     48          LDA  VBLOFF     (4) BACK TO VBL YET?
1F3F:EA           49          NOP             ;(2)
1F40:30 EE        50          BMI  LP17029    (3,2) NO; SLEW BACK
1F42:             51 *
1F42:A2 05        52          LDX  #5         (2)  YES; END VBL IS PRECISELY LOCATED
```

Figure 3.12 Assembler Listing: Locating VBL Precisely (1 of 2).

```
1F44:20 84 1F   53          JSR  WAITX1K   (5000) NOW WAIT 5755 CYLES FOR TEXT WINDOW
1F47:A0 49      54          LDY  #73        (2)
1F49:20 7A 1F   55          JSR  WAITX10    (730)
1F4C:48         56          PHA             ;(3)
1F4D:68         57          PLA             ;(4)
1F4E:AD FF FF   58          LDA  $FFFF      (4)
1F51:A2 08      59          LDX  #8         (2)
1F53:AD 51 C0   60 TXTTIME  LDA  TEXT       (4)
1F56:20 8D 1F   61          JSR  RTS1       (12) WINDOW RIGHT = WINDOW LEFT + 21
1F59:A5 00      62          LDA  $00        (3)
1F5B:EA         63          NOP             ;(2)
1F5C:AD 50 C0   64          LDA  GRAFIX     (4)
1F5F:A0 03      65          LDY  #3         (2) WINDOW LEFT = WINDOW RIGHT + 44
1F61:20 7A 1F   66          JSR  WAITX10    (30)
1F64:A5 00      67          LDA  $00        (3)
1F66:CA         68          DEX             ;(2)
1F67:D0 EA      69          BNE  TXTTIME    (3,2) SWITCHING TIME = 8 X 65 - 1 = 519
1F69:A2 10      70          LDX  #16        (2) WAIT 17030 - 519 = 16511
1F6B:20 84 1F   71          JSR  WAITX1K    (16000) BEFORE WINDOW LEFT
1F6E:A0 32      72          LDY  #50        (2)
1F70:20 7A 1F   73          JSR  WAITX10    (500)
1F73:A2 08      74          LDX  #8         (2)
1F75:EA         75          NOP             ;(2)
1F76:D0 DB      76          BNE  TXTTIME    (3)
1F78:           77 *
1F78:D0 01      78 LOOP10   BNE  SKIP
1F7A:88         79 WAITX10  DEY             WAIT Y-REG TIMES 10
1F7B:88         80 SKIP     DEY
1F7C:EA         81          NOP
1F7D:D0 F9      82          BNE  LOOP10
1F7F:60         83          RTS
1F80:48         84 LOOP1K   PHA
1F81:68         85          PLA
1F82:EA         86          NOP
1F83:EA         87          NOP
1F84:A0 62      88 WAITX1K  LDY  #98        WAIT X-REG TIMES 1000
1F86:20 7A 1F   89          JSR  WAITX10
1F89:EA         90          NOP
1F8A:CA         91          DEX
1F8B:D0 F3      92          BNE  LOOP1K
1F8D:60         93 RTS1     RTS
1F8E:           94 *
1F8E:00         95 MSG      DFB  $00        SWITCH IN THE BLACK
1F8F:AA CC E9   96          ASC  "*Little   Text Window* "
1F92:F4 F4 EC
1F95:E5 A0 D4
1F98:E5 F8 F4
1F9B:A0 D7 E9
1F9E:EE E4 EF
1FA1:F7 AA A0

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Figure 3.12 Assembler Listing: Locating VBL Precisely (2 of 2).

SOFTWARE APPLICATION

APPLE TIMING LOOPS

It is not generally known that the 6502 clock of the Apple is not fixed frequency, and there is some confusion about what that frequency is. This is not important in most Apple programs, but the frequency and stability of the MPU clock are important factors in precision timed loop assembly language programs.

The *Apple II Reference Manual for IIe Only* inaccurately gives the PHASE 0 frequency as 1.022727 MHz. This is 14.31818 divided by 14, and it would be the PHASE 0 frequency if there was no long cycle. The composite frequency of the Apple is 1.02048432 MHz, which is 14.31818 x (65/(65 x 14 + 2)). The average period of duration of an Apple 6502 machine cycle is .9799268644 microseconds. This is the value which should be used for computing exact time durations of Apple programs. In PAL-based Apple IIe's with 14.25045 MHz oscillators, the average machine cycle duration is .9845842925 microseconds. In future PAL-based Apple IIe's with the 14.25 MHz hybrid oscillator, the average machine cycle duration will be .9846153846 microseconds.

When very precise time measurement is necessary, the programmer has to consider the impact of clockpulse jitter, which is caused by the long cycle. Since the Apple IIe has no real time clock, timed output must be done with program loops which take a specific number of clock pulses to execute. When possible, these loops should be written in multiples of 65 cycles. This will eliminate loop output jitter. Otherwise the application must be able to tolerate a 140-nanosecond jitter. 140 nanoseconds is the difference between a normal cycle and a long cycle. The programmer should be aware of Apple clockpulse jitter and determine its affect on his particular application.

HARDWARE APPLICATION

AN APPLESOFT EMULATOR FOR THE TIMING HAL

Analyzing the Apple IIe HAL timing outputs can be pretty difficult, especially when you begin looking at the LDPS' and VID7M variations. Figure 3.13 is an Applesoft program which draws timing diagrams of the HAL outputs based on logic equations like those used to specify a HAL/PAL program. The program lets you vary the SEGB, gated GR+2', CASEN', 80COL', and VID7 inputs to the HAL and see the resulting timing diagram plotted out on the HIRES screen.

Figure 3.14 shows two sample timing diagrams plotted by the HAL emulator. The plotting always starts with the signals in the states shown at the left of Figure 3.14, and H0 always stays low for two counts after the first time it falls. This results in the plotting of the long cycle. If you initialize the *DOS TOOLKIT* HRCG program before running the emulator, the names of the signals will be drawn on the left side as shown in Figure 3.14.

The HAL emulator can also serve as a design aid for those persons interested in experimenting with alternate timing schemes for the Apple IIe. By changing any of the equation definitions in lines 2000—2470, you can check out how the timing signals would look if the HAL were programmed differently. Also when you run the emulator, it allows you to specify scanning instead of plotting. If you select scanning, the emulator will scan through all possible starting states of the HAL outputs, excluding LDPS' and VID7M. For each initial setting, the emulator prints the number of 14M cycles before the outputs reach the states pictured at the left in Figure 3.14. This verifies that a given set of HAL equations cannot cause the timing chain to hang in some invalid sequence. It takes several hours for the emulator to scan all the possibilities for a set of equations, so turn on your printer and be prepared to wait if you decide to perform a complete scan.

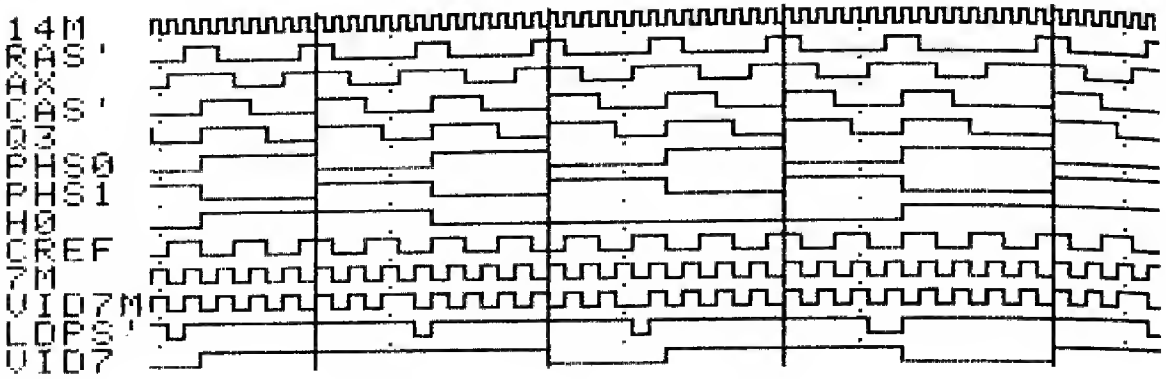
An interesting design problem that some readers might wish to tackle is the right side cutoff of HIRES delayed video. LDPS' and VID7M logic equations are such that the last video cycle is always cut off after Q3' • AX' (see PHASE 1 during the long cycle). It would be preferable if this cutoff was delayed by one 14M period when the last video cycle is HIRES delayed because, as things are, you cannot plot orange dots at the far right of the display in HIRES40 mode. I grappled with this problem and was unable to come up with a working set of HAL logic equations that would solve it, not even if the video generator video ROM was programmed so that HIRES bit 7 mirrored HIRES bit 6. I finally gave up on the problem although I wouldn't pronounce it unsolvable. Perhaps a reader more resourceful than I can work it out. No fair rewiring the motherboard or switching to a 350-nanosecond video ROM.

```
100  REM
110  REM
120  REM                APPLE IIE HAL/PAL TIMING EMULATOR
130  REM
140  REM                     BY JIM SATHER   2/14/84
150  REM
160  REM
200  REM  ********** INITIAL SIGNAL DATA
210  DATA 0,0,0,1,0,1,0,0,0,0,1,0
215  DIM W(11,8): DIM V(8): DIM S(8): DIM I(11)
220  FOR SIGNAL = 0 TO 11: READ W(SIGNAL,0):I(SIGNAL) = W(SIGNAL,0): NEXT
230  REM
240  REM  ********** TITLES
250  DATA "14M  ","RAS' ","AX   ","CAS' ","Q3   ","PHS0 ","PHS1"
255  DATA "H0   ","CREF ","7M   ","VID7M","LDPS'","VID7 "
260  DIM TITLES$(12): FOR SIGNAL = 0 TO 12: READ TITLE$(SIGNAL): NEXT
270  REM
1000  REM  ********  CLEAR SCREEN AND PREPARE TO DRAW WAVEFORMS
1030  TEXT : HOME :HCNT = 0:VCNT = 0
1032  PRINT "ENTER P (PLOT) OR S (SCAN)": GET B$: HOME : IF B$ = "S" THEN  GOTO 4000
1040  PRINT "SEGB="SEGB"  OK? Y/N": GET B$: IF B$ <  > "Y" THEN SEGB =  NOT SEGB: GOTO 1040
1050  PRINT "GR'="GX%"  OK?": GET B$: IF B$ <  > "Y" THEN GX% =  NOT GX%: GOTO 1050
1060  PRINT "CASEN'="CSEN%"  OK?": GET B$: IF B$ <  > "Y" THEN CSEN% =  NOT CSEN%: GOTO 1060
1070  PRINT "80COL'="COL80%"  OK?": GET B$: IF B$ <  > "Y" THEN COL80% =  NOT COL80%: GOTO 1070
1080  PRINT "ENTER VID7 STATES ";: FOR A = 1 TO 7: PRINT V(X)",";: NEXT : PRINT V(8);: HTAB 18
1090  INPUT V(1),V(2),V(3),V(4),V(5),V(6),V(7),V(8): PRINT : VTAB 21:HCNT = 0:VCNT = 0
1094  PRINT "SEGB="SEGB"   GR'="GX%"   CASEN'="CSEN%"   80COL'="COL80%
1096  PRINT : PRINT "VID7=";: FOR A = 1 TO 8: PRINT V(A);: NEXT
1098  VTAB 1: HGR : HCOLOR= 3: PRINT  CHR$ (17);: REM CTRL-Q HOMES CURSOR
1099  FOR SIGNAL = 0 TO 12: PRINT TITLE$(SIGNAL): NEXT
1100  REM
1101  REM  ********** PLOT LEFT TO RIGHT FOR/NEXT LOOP
1105  FOR X = 36 TO 276 STEP 4: HPLOT X,5 TO X,1 TO X + 2,1 TO X + 2,5 TO X + 3,5
1120  FOR SIGNAL = 0 TO 11: FOR TERM = 1 TO 8:W(SIGNAL,TERM) = 0: NEXT : NEXT
1130 RAS% = W(0,0):AX = W(1,0):CAS% = W(2,0):Q3 = W(3,0):P0 = W(4,0):P1 = W(5,0)
1140 H0 = W(6,0):CREF = W(7,0):S7M = W(8,0):V7M = W(9,0):LDPS% = W(10,0):VID7 = W(11,0)
2000  REM
2010  REM  ********** BEGIN EQUATION DEFINITIONS
2020  REM
2030  REM  AVAILABLE INPUTS ---> S7M,CREF,H0,VID7,SEGB,GX%,CSEN%,VD80%
2040  REM           OUTPUTS ---> RAS%,AX,CAS%,Q3,P0,P1,V7M,LDPS%
2060  REM                        % IS TAG FOR ACTIVE LOW SIGNALS LIKE CASEN'
2070  REM
2080  REM  ********** RAS' (RAS%)
2090 W(0,1) = Q3
2100 W(0,2) =  NOT RAS% AND  NOT AX
2110 W(0,3) =  NOT RAS% AND CREF AND H0 AND P0
2120 W(0,4) =  NOT RAS% AND  NOT S7M AND H0 AND P0
2130  REM  ********** AX
2140 W(1,1) =  NOT RAS% AND Q3
2150 W(1,2) =  NOT AX AND Q3
2160  REM  ********** CAS' (CAS%)
2170 W(2,1) =  NOT AX AND  NOT CSEN%
2180 W(2,2) =  NOT AX AND P1
2190 W(2,3) =  NOT CAS% AND  NOT RAS%
2200  REM  ********** Q3
2210 W(3,1) =  NOT AX AND P1 AND  NOT S7M
2220 W(3,2) =  NOT AX AND P0 AND S7M
2230 W(3,3) =  NOT Q3 AND  NOT RAS%
2240  REM  ********** PHASE 0 (P0)
2250 W(4,1) = P0 AND RAS% AND  NOT Q3
2260 W(4,2) =  NOT P0 AND  NOT RAS%
2270 W(4,3) =  NOT P0 AND Q3
```

Figure 3.13 BASIC Listing: An Apple IIe Timing HAL Emulator (1 of 2).

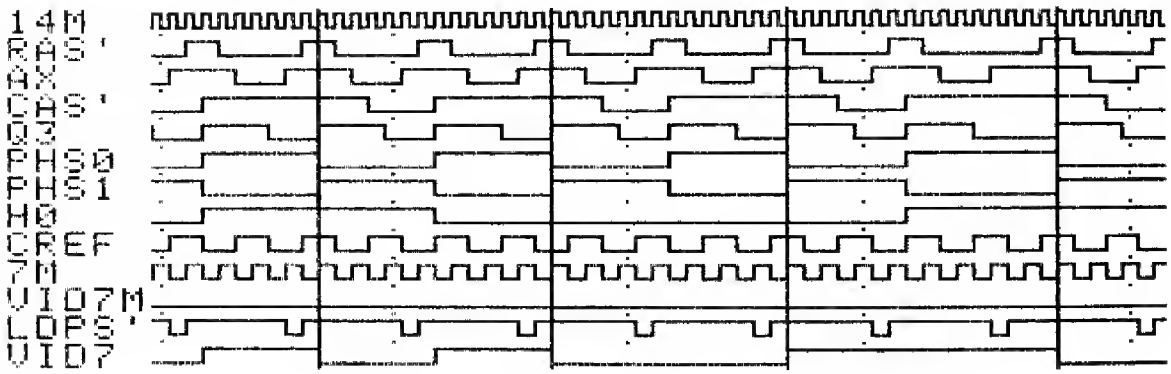
```
2280  REM  ********** PHASE 1 (P1)
2290 W(5,1) =  NOT P0 AND RAS% AND  NOT Q3
2300 W(5,2) = P0 AND  NOT RAS%
2310 W(5,3) = P0 AND Q3
2320  REM  ********** VID7M (V7M)
2330 W(9,1) =  NOT GX% AND SEGB
2340 W(9,2) = GX% AND  NOT COL80%
2350 W(9,3) = GX% AND S7M
2360 W(9,4) =  NOT VID7 AND P1 AND  NOT Q3 AND  NOT AX
2370 W(9,5) =  NOT H0 AND CREF AND P1 AND  NOT Q3 AND  NOT AX
2380 W(9,6) = V7M AND AX
2390 W(9,7) = V7M AND P0
2400 W(9,8) = V7M AND Q3
2410  REM  ********** LDPS' (LDPS%)
2420 W(10,1) =  NOT Q3 AND  NOT AX AND  NOT COL80% AND GX%
2430 W(10,2) =  NOT Q3 AND  NOT AX AND P1 AND GX%
2440 W(10,3) =  NOT Q3 AND  NOT AX AND P1 AND SEGB
2450 W(10,4) =  NOT Q3 AND  NOT AX AND P1 AND  NOT VID7
2460 W(10,5) =  NOT Q3 AND  NOT AX AND P1 AND CREF AND  NOT H0
2470 W(10,6) =  NOT Q3 AND AX AND  NOT RAS% AND P1 AND VID7 AND  NOT SEGB AND  NOT GX%
3000  REM
3020  REM  ********** THE FOLLOWING DEFINITIONS ARE EXTERNAL TO THE HAL/PAL.
3040  REM  ********** H0
3045  IF  NOT RAS% OR  NOT P1 OR Q3 THEN 3060
3050 HCNT = HCNT + 1: IF HCNT <  > 3 THEN W(6,1) =  NOT H0: GOTO 3070
3060 W(6,1) = H0: GOTO 3080
3070  REM  ********** CREF
3080  IF S7M THEN W(7,1) =  NOT CREF
3090  IF  NOT S7M THEN W(7,1) = CREF
3100  REM  ********** 7M (S7M)
3110 W(8,1) =  NOT S7M
3120  REM  ********* VID7
3130  IF RAS% AND  NOT Q3 THEN VCNT = VCNT + 1: IF VCNT < 9 THEN W(11,1) = V(VCNT): GOTO 3150
3140 W(11,1) = VID7
3150  REM
3160  REM  ********** DEFINITIONS NOW COMPLETE
3170  REM  NOW "OR" ALL THE TERMS FOR EACH SIGNAL AND DRAW THE SIGNALS.
3180  REM
3190  IF  NOT SCAN AND P0 AND  NOT Q3 AND RAS% THEN  HPLOT X,0 TO X,102: REM REFERENCE LINES
3195  IF  NOT SCAN AND P1 AND  NOT Q3 AND  NOT AX THEN  FOR Y = 7 TO 95 STEP 8: HPLOT X - 2,Y: NEXT
3200  FOR SIGNAL = 0 TO 11:Y = SIGNAL * 8 + 13: FOR TERM = 8 TO 2 STEP  - 1
3210 W(SIGNAL,TERM - 1) = W(SIGNAL,TERM - 1) OR W(SIGNAL,TERM): NEXT TERM
3215  IF SIGNAL < 6 OR SIGNAL = 9 OR SIGNAL = 10 THEN W(SIGNAL,1) =  NOT W(SIGNAL,1)
3217  IF SCAN THEN  NEXT SIGNAL: RETURN
3220  HPLOT X,Y - 4 * W(SIGNAL,0) TO X,Y - 4 * W(SIGNAL,1) TO X + 3,Y - 4 * W(SIGNAL,1)
3230 W(SIGNAL,0) = W(SIGNAL,1): NEXT SIGNAL: NEXT X
3240  PRINT  CHR$ (4);"PR#2": END
4000  REM ********************************************************************************
4005  REM
4010  REM  SCAN ALL POSSIBLE INITIAL CONDITIONS TO MAKE
4020  REM  CERTAIN PAL SYNCS UP CORRECTLY.
4030  REM
4050 SCAN = 1: FOR SIGNAL = 1 TO 7: PRINT TITLE$(SIGNAL);: NEXT : PRINT "  X": POKE 34,1
4060  FOR SIGNAL = 0 TO 8:W(SIGNAL,0) = S(SIGNAL)
4070  IF SIGNAL < 7 THEN  HTAB SIGNAL * 5 + 2: PRINT S(SIGNAL);
4080  NEXT SIGNAL
4090  FOR X = 1 TO 100:HCNT = 0:CFLAG = 0: GOSUB 1120: REM UPDATE SIGNALS
4100  FOR SIGNAL = 0 TO 8:W(SIGNAL,0) = W(SIGNAL,1): IF W(SIGNAL,0) <  > I(SIGNAL) THEN CFLAG = 1
4120  NEXT SIGNAL: IF CFLAG THEN  NEXT X
4140  PRINT "     "X:SIGNAL = 7: IF X = 100 THEN  PRINT  CHR$ (7) CHR$ (7) CHR$ (7);: GET B$
4160 SIGNAL = SIGNAL - 1
4165  IF SIGNAL < 0 THEN  PRINT  CHR$ (7)"ALL POSSIBILITIES SCANNED": POKE 34,0: END
4170 S(SIGNAL) =  NOT S(SIGNAL): IF S(SIGNAL) = 0 THEN  GOTO 4160
4180  GOTO 4060
```

Figure 3.13 BASIC Listing: An Apple IIe Timing HAL Emulator (2 of 2).



SEGB=0 GR'=0 CASEN'=0 80COL'=0

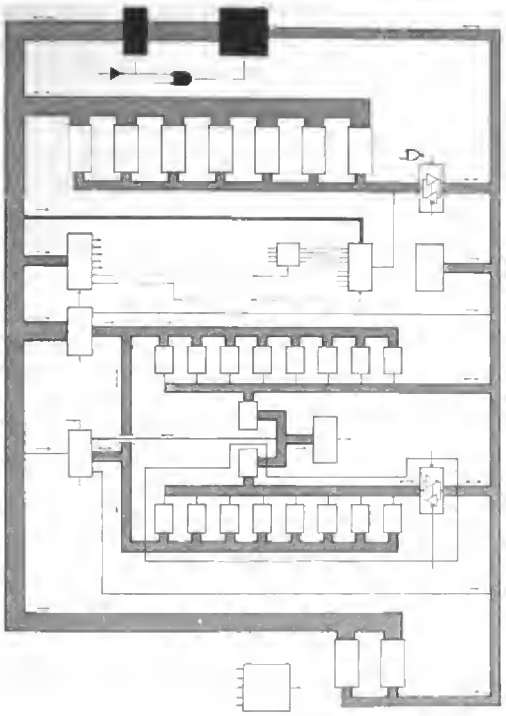
VID7=11101101



SEGB=0 GR'=1 CASEN'=1 80COL'=0

VID7=10100110

Figure 3.14 HIRES Dumps from the Timing HAL Emulator.



chapter 4

The 6502 Microprocessor

The 6502 MPU (Micro Processing Unit) is the device in the Apple IIe which executes stored sequential programs. It is a single 40-pin integrated circuit that executes 6502 machine language programs as it reads them from the data bus. It can be thought of as the brains of the Apple.

The 6502 was designed by MOS Technology in the mid 1970s as part of their MCS6500 series microprocessor family. It has been a popular choice as a microprocessor for personal computers, being used in computers produced by Apple, Atari, Commodore, Ohio Scientific, Rockwell International, and other manufacturers. The 6502 gives adequate computing speed and versatility at a very low cost. Its programming language is very simple, making it an ideal MPU for the occasional computer programmer.

The most important 6502 related knowledge for an Apple owner to attain is programming knowledge. The ability to read and write 6502 assembly language programs greatly expands the horizons of an Apple computerist. 6502 assembly language is not, however, a major topic of this book. These pages are concerned primarily with the hardware implementation of the 6502 in the Apple IIe computer. Volumes have been written about various aspects of the 6502, especially programming. The choice of 6502 topics in this chapter was governed by the unique features of 6502 use in the Apple, and by the goal of this book to fill information gaps in Apple literature available to the public.

Manufacturers of the 6502 are:

MOS Technology, Inc.
950 Rittenhouse Rd.
Norristown, Pa. 19403

Synertek, Inc.
P.O. Box 552
Santa Clara, Ca. 95052

Rockwell International
Microelectronic Devices
P.O. Box 3669, RC55
Anaheim, Ca. 92803

6502 SIGNALS

There are 40 pins on a 6502, three of which serve no function. In addition to the address output and data input/output, there are four outputs (R/W', PHASE 1, PHASE 2, and SYNC) and six inputs (READY, IRQ', NMI', PHASE 0, SET OVERFLOW', and RESET'). There are three power supply connections. One pin requires +5 volts and two pins require ground. Figure 4.1 shows the 6502 pin assignments. Figure 4.2 shows the 6502 hardware implementation in the Apple. A brief discussion of the 6502 signals with Apple implementation notes follows.

Clockpulses—PHASE 0, PHASE 1, PHASE 2

The 6502 has most of its required clockpulse generation circuitry built-in. It requires only an externally generated time base which can be implemented in several ways. In the Apple, the PHASE 0 time base is developed independent of 6502 internal circuits and fed as the clockpulse input to the 6502.

The 6502 generates its required PHASE 1 and PHASE 2 clocks from the PHASE 0 input. PHASE 1 is high during the first half of a machine cycle, and PHASE 2 is high during the second half of a machine cycle. PHASE 1 is not the simple inversion of PHASE 2. There is a slight delay between the PHASE 1 transitions and the PHASE 2 transitions. The rising edge of one always follows the falling edge of the other. The PHASE 1 and PHASE 2 clocks are available at pins 3 and 39 of the 6502. PHASE 1 and PHASE 2 are not connected in the Apple IIe but are used only inside the 6502.

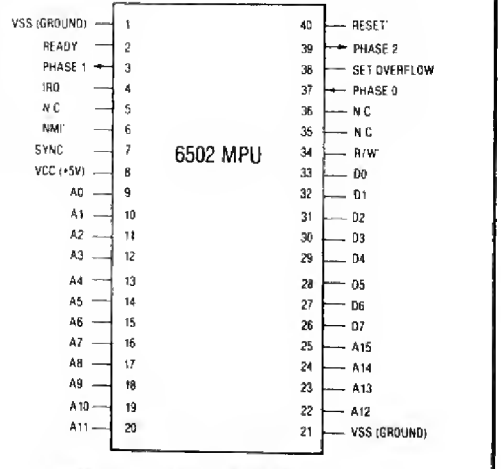


Figure 4.1 6502 Pin Assignments.

Address and R/W'

During every machine cycle, the 6502 places an address on its address output. In association with the address it outputs, it brings its R/W' line high or low, thereby telling the world whether it wants to read or write data. With 16 address lines, the 6502 is capable of producing 65536 different values at its address output.

The 6502 address and R/W' outputs are not tri-state in the Apple, but these signals are connected to the address bus through external **tri-state bus drivers**. This enables peripheral cards to gain access to the address bus via the DMA' line.

Data Bus

The data input/output of the 6502 is eight lines. This gives the 6502 its overall classification as an 8-bit microprocessor. Data direction is inward except during PHASE 2 of write cycles. In the Apple IIe, the 6502 data lines are connected directly to the data bus.

RESET'

The RESET' input to the 6502 causes the 6502 to start or restart. A RESET' causes the 6502 to disable interrupts and begin program execution at an address stored in locations $FFFC and $FFFD of ROM. 6502 operation is inhibited while RESET' is held low. The RESET' sequence begins when RESET' transits from low to high.

In the Apple IIe, the RESET' line is connected to pin 31 of the peripheral slots, to the keyboard RESET key, and to pin 15 of the IOU. A peripheral card can cause RESET' to drop, respond to RESET', or ignore RESET'. The IOU responds to RESET', but it also causes RESET' to drop when power is first applied to the computer.

Interrupts—IRQ' and NMI'

The interrupts cause the 6502 to stop its sequential program execution and execute interrupt handling routines. Interrupts are normally associated with input/output functions, but they are a way for any type of device to get the microprocessor's attention. The IRQ' (Interrupt ReQuest) is enabled or disabled by program control, so the 6502 doesn't have to respond to an IRQ'. The NMI' (Non-Maskable Interrupt) cannot be disabled by program control.

An NMI' causes the 6502 to perform an interrupt sequence after the current 6502 instruction has been executed. The 6502 saves its program location counter and its Status Register (with BREAK flag reset)

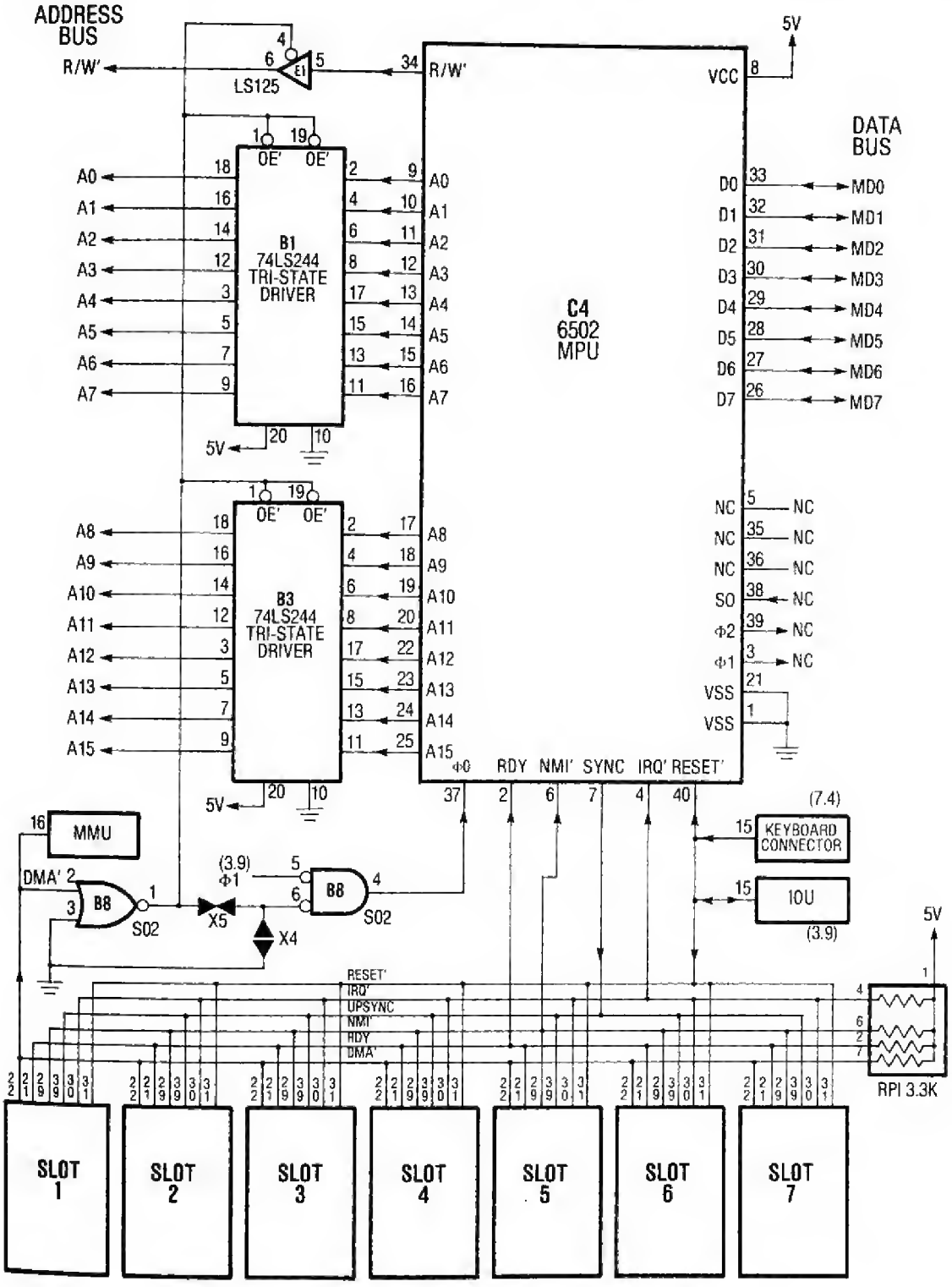


Figure 4.2 Schematic: Apple IIe 6502 Connections.

in an area of RAM called the stack. It disables interrupt requests, then begins program execution at the address stored in $FFFA and $FFFB of memory. The NMI' input to the 6502 is edge sensitive, meaning the 6502 responds only to a high to low transition of NMI'. To generate a second interrupt, the NMI' line must be brought high, then low again.

An IRQ' causes the 6502 to perform an interrupt sequence after the current instruction has been executed if the program has interrupt requests enabled. The IRQ' sequence is identical to the NMI' sequence, except that the address of the IRQ' handling routine is stored at $FFFE and $FFFF. The IRQ' signal is not edge sensitive, so the IRQ' must go high before interrupts are enabled again, or the same interrupt will be answered more than once.

The interrupt inputs to the 6502 are connected to the peripheral slots in the Apple. There are no motherboard devices which generate 6502 interrupts, and I/O in the Apple IIe is normally accomplished without interrupts. The Apple IIe mouse is IRQ' based. Most real time clock cards are capable of generating interrupt requests, and cards which dump Apple memory to disk are based on non-maskable interrupts. The IRQ' line is tied to pin 30 of the peripheral slots, and the NMI' line is tied to pin 29.

READY

Bringing the READY input to the 6502 low during the PHASE 1 clock in a read cycle causes the 6502 to go into its wait state. In the wait state, the 6502 holds the current address and does nothing. The wait state lasts until READY is sensed high during PHASE 2. If the high to low transition occurs during a write cycle, the wait state will not begin until the next read cycle.

The wait state of the 6502 can be used for interfacing to slow memories, single step operation, slow step operation, or just plain stopping the MPU indefinitely. It has no impact on the Apple's video circuitry or on RAM refresh, so the video display appears frozen on the screen when the 6502 is halted via the READY line. The READY line in the Apple is connected only to pin 21 of the peripheral slots. This 6502 capability has gone largely unexploited in the Apple.

SYNC

The 6502 SYNC output goes high when the 6502 is performing an op code fetch. This is the first cycle in the execution of any instruction in which the 6502 fetches the 1-byte operational code of the instruction. The SYNC signal can be used for single instruction execution steps (in conjunction with the READY line) and otherwise identifying the op code of a 6502 instruction. The SYNC output of the 6502 is connected only to pin 39 of the peripheral slots in the Apple IIe.

SET OVERFLOW'

A high to low transition on the SET OVERFLOW' line sets the overflow flag of the 6502. The overflow flag is normally set or reset as a logical result of some 6502 instructions, but the SET OVERFLOW' input forces the flag regardless of instruction execution.

The SET OVERFLOW' input has limited value as a control input, because it must be used only in conjunction with instructions that affect the overflow flag or in avoidance of such instructions so as not to interfere with them. It is not connected in the Apple IIe.

6502 CONNECTIONS IN THE APPLE IIe

Figure 4.2 shows the 6502 hardware implementation in the Apple IIe. R/W' connections are routed to the address bus through external drivers, and data lines are connected directly to the data bus. The PHASE 0 clock is PHASE 1 from the timing generator, inverted and gated by DMA' from the peripheral slots. All other signals are connected directly to the peripheral slots.

The method of tying the 6502 control inputs to multiple sources is called **wire-ORing** or **collector-ORing**. A logical OR function is achieved by tying lines directly together. As an example, if Slot 0 OR Slot 1 OR any other slot pulls pin 29 low, the 6502 will sense a non-maskable interrupt. In a wire-OR circuit, the line is pulled high by a voltage through a resistor if no card is pulling the line low. Peripheral cards should not try to pull the wire-OR lines high. They either pull the line low or present a high impedance to the line, usually by driving the line with open collector TTL circuits. The 6502 literature specifies that 3000 ohm pull-up resistors be used for wire-OR inputs to the 6502, and the Apple IIe uses a 3300 ohm SIP (Single In line Package) resistor for this function.

The tri-state address bus driver is necessary for DMA operations because the 6502 address and R/W' connections are not tri-state. The address drivers used in the Apple are LS244 8-bit tri-state bus drivers for the address lines and one fourth of an LS125 for R/W'.

6502 MEMORY USAGE

Use of the 6502 in the Apple dictates various aspects of the memory layout. For example, addresses $0—$1FF are always RAM in a 6502 system. Apart from design dictates, the 6502 also uses parts of memory so that they are normally not available for Apple programs.

Page 0 and Page 1 ($0—$FF and $100—$1FF) of a 6502 system must be RAM simply because the 6502 has special read/write uses for Page 0 and Page 1. **Page 0** locations are used as **indirect address locations** in 6502 machine language. Additionally, the 6502 has a **zero page addressing mode** which speeds and compacts programs making heavy use of zero page locations for various storage functions. As a result, big machine language programs like Applesoft BASIC make heavy use of zero page locations. If BASIC is operating and you indiscriminately POKE values into zero page locations, you will deep six BASIC. This is because the critical pointers of BASIC will be lost. The following program must crash:

```
10  FOR A = 0 TO 255 :
    POKE A,0 : NEXT A : END
```

Page 1 is the 6502 stack. The stack of a microprocessor is an area of RAM which it uses as a **last in-first out memory**. To the computer program, the stack is like a stack of playing cards which it can discard to or draw from. Conceptually, data is stored to the top of the stack or withdrawn from the top of the stack. The stack is actually part of RAM. While the program pushes data to or pulls data from the stack, the MPU must increment or decrement a read address and keep track of where in memory the "top" of the stack is. In the 6502, the location of the "top" of the stack is stored internally in an 8-bit register called the **Stack Pointer**. When the stack is accessed, the 6502 addresses a location in Page 1 of RAM determined by the Stack Pointer. Virtually all machine language programs access the stack via Jump SubRoutine and ReTurn from Subroutine instructions, so at no time can a program indiscriminately modify Page 1.* The following BASIC program will crash as surely as the earlier one:

```
10  FOR A = 256 TO 511 :
    POKE A,0 : NEXT A : END
```

*Exceptions are copy protection schemes which call for programming without JSR, RTS, PHP, PLP, PHA, or PLA instructions. In these schemes, critical data is stored in Page 1 of memory, and most attempts to examine memory result in the loss of the critical Page 1 data.

The 6502 also dictates that the highest memory location is $FFFF, and that it will be assigned to ROM. That $FFFF is the highest address is an obvious consequence of the fact that the 6502 has 16 address lines. In a similar vein, the eight data lines of the 6502 dictate that memory is organized into 8-bit locations. The reason for assigning the highest address to ROM is that the 6502 RESET, NMI, and IRQ vectors must be stored in locations $FFFA through $FFFF. In particular, the RESET vector in ROM enables the Apple to immediately begin executing a non-erasable program at power up.

Since the 6502 has no special input/output control features, it must control input/output functions with commands decoded from the address bus. In the Apple, addresses are assigned to the peripheral slots and built-in I/O functions which could be otherwise assigned to memory. This is referred to as **memory mapped I/O**. It was logical in the Apple design to assign the address space between RAM and ROM to I/O. That way there are three contiguous addressing groups: RAM ($0—$BFFF), I/O ($C000—$CFFF), and ROM ($D000—$FFFF).

6502 TIMING IN THE APPLE IIe

The 6502 was designed to be similar to the Motorola MC6800 microprocessor, but improved. The clock requirements of the 6502 are the same as the MC6800—two alternating positive pulses. In the MC6800, the two clocks must be generated externally and input. In the 6502, the two clocks are generated internally from the PHASE 0 clock input. This is one of the 6502 improvements.

The relationship between the PHASE 0 clock input and the PHASE 1 and PHASE 2 6502 clocks is shown in Figure 4.3. The PHASE 1 and PHASE 2 clocks are not symmetrical but are low slightly longer than they are high. The high period of one clock always fits neatly inside the low period of the other. The PHASE 1 and PHASE 2 transitions are clocked by the transitions of the PHASE 0 input in a repetitive cycle. The falling edge of PHASE 0 is followed by the falling edge of PHASE 2 and then the rising edge of PHASE 1. The rising edge of PHASE 0 is followed by the falling edge of PHASE 1 and the rising edge of PHASE 2. To put it differently, PHASE 0 falling clocks the end of PHASE 2 then the beginning of PHASE 1, and PHASE 0 rising clocks the end of PHASE 1 then the beginning of PHASE 2.

The effect of the **long cycle** on 6502 clocks is to elongate PHASE 2 by 140 nanoseconds. This has no

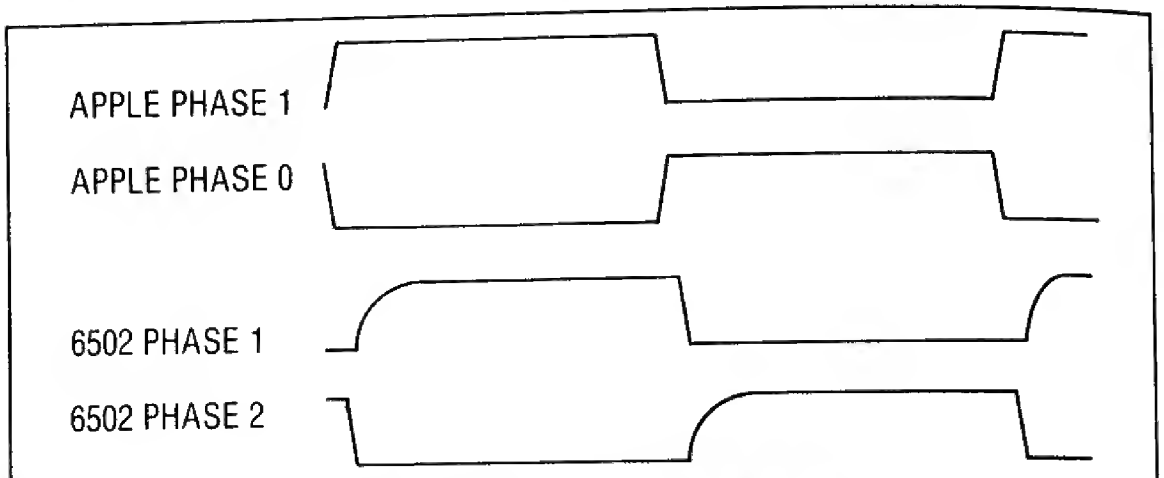


Figure 4.3 6502 Clockpulse Relationships.

particular ill effects outside of program timing considerations mentioned in the previous chapter. By lengthening PHASE 2, all response criteria for communicating with the 6502 become less critical. The following timing discussions are valid for either a normal cycle or a long cycle, but the diagrams picture normal cycles. The timing specifications of the 6502 are not affected by the long cycle.

The 6502 PHASE 1 clock is not the same as the PHASE 1 signal developed in the timing generator. PHASE 1 from the timing generator is simply PHASE 0 inverted. It was named PHASE 1 because of its kinship with the 6502 PHASE 1 clock. Semantic ambiguity is a great way to confuse those who would understand. The term which is distributed to the peripheral slots, address decode, and RAM is PHASE 1 from the timing generator. The 6502 PHASE 1 clock is used only inside the 6502. In this chapter only, "PHASE 1" refers to the 6502 PHASE 1 clock. Outside of this chapter "PHASE 1" refers to the inversion of PHASE 0, distributed from the timing generator.

The Apple IIe uses a 6502A which has less critical timing specifications than the 6502 (no designation letter) that was used in the Apple II. The 6502A is rated for use in 2 MHz computers, but Apple uses the 6502A in their 1 MHz computer to provide greater error margins in Apple IIe timing.*

*The *Apple II Reference Manual for IIe Only* says that the IIe uses a 6502B, and it shows 6502B timing specs in the MPU timing diagram. Also, the MPU socket on my motherboard is labeled 6502B. But Apple applications engineer Peter Baum informed me that it was all a mistake, and that only 2 MHz 6502s (6502As) were used in the Apple IIe. Peter also said that the reason for using 6502As instead of 1 MHz 6502s is that error margins are increased at a cost of only $0.25 per 6502.

Timing specifications in the 6502 are referenced to the rising and falling edge of the PHASE 2 clock (at the .4V point). Important timing specifications for Synertek 6502As are shown here, with Mos Technology and Rockwell International ratings shown in parenthesis when they differ:

1. The 6502 address and R/W' line will be valid within 140 nanoseconds (150 nsec Mos Technology) after the falling edge of PHASE 2. They will stay valid until at least 30 nanoseconds after the next falling edge of PHASE 2. The address becomes valid during the first part of PHASE 1.
2. 6502 write data will be valid within 100 nanoseconds after the rising edge of PHASE 2. The write data will remain valid until at least 60 nanoseconds (30 nsec Mos Technology; 30 nsec Rockwell) after the falling edge of PHASE 2.
3. 6502 read data must be valid at least 50 nanoseconds (40 nsec Rockwell) before the falling edge of PHASE 2 and must be held valid at least 10 nanoseconds after the falling edge of PHASE 2. PHASE 2 falling is the 6502 data transfer clock.
4. The maximum delay between PHASE 0 falling and PHASE 2 falling is 65 nanoseconds. The maximum delay between PHASE 0 rising and PHASE 2 rising is 75 nanoseconds. These values are specified only by Synertek and only with a 100-picofarad load on PHASE 2.

The time periods represent worst case conditions over an operating range from 0 to 70 degrees centigrade. Worst case timing specifications are shown in Figure 4.4. Synertek time values are used because

that seems to be the brand used in the Apple IIe, and because Synertek values are used in the timing specification given by Apple in the IIe reference manual.* Also, only Synertek publishes a specification for the important PHASE 0 to PHASE 2 delay. Ten nanoseconds could probably be subtracted from the clockpulse delay specifications to reflect the fact that there is no load on PHASE 2 in the Apple.

The Synertek, MOS Technology, and Rockwell International 6502s are probably all made the same. When one company gives tighter specifications than another, it obligates itself to test its microprocessors using more difficult criteria.

Figure 4.5 is a diagram showing the timing relationships actually found in one Apple. The measurements were made in an Apple IIe using a Synertek 6502 marked 8307 (January 7, 1983?), S10891, 370-6502 (no letter designator?). Figure 4.5 may be considered fairly typical of 6502 timing in the Apple IIe, but one must be wary of an experiment with only one sample. The important features of Figure 4.5 are:

1. PHASE 1 and PHASE 2 transitions occur roughly 30 nanoseconds after PHASE 0 falling at the peripheral slots. Delays measured from PHASE 0 rising are longer because the 74S02 clockpulse gate takes longer to bring its output high than it does to bring it low (see Figure 4.2).
2. The 6502 address becomes valid at the address bus 124 nanoseconds after PHASE 0 falls at the peripheral slots. This indicates a setup time of under 100 nanoseconds from PHASE 2 falling. Using the worst case conditions for a 6502A, the 6502 address in an Apple IIe will always be valid at the address bus before Q3 falls.
3. Write data becomes valid at the data bus 108 nanoseconds after PHASE 0 rises. With a 42-nsec delay from PHASE 0 rising to PHASE 2 rising, this indicates a write data setup time of 66 nsec with the Apple in a normal room temperature environment. 6502 write data must be valid before CAS' falls for it to be read by

*See Figure 7-1 on page 142 of the *Apple II Reference Manual for IIe Only*. The reference manual timing chart shows Synertek 6502B specifications, and it is inaccurate since it shows the specifications referenced to PHASE 0 falling. All 6502 specifications are referenced to PHASE 2 falling.

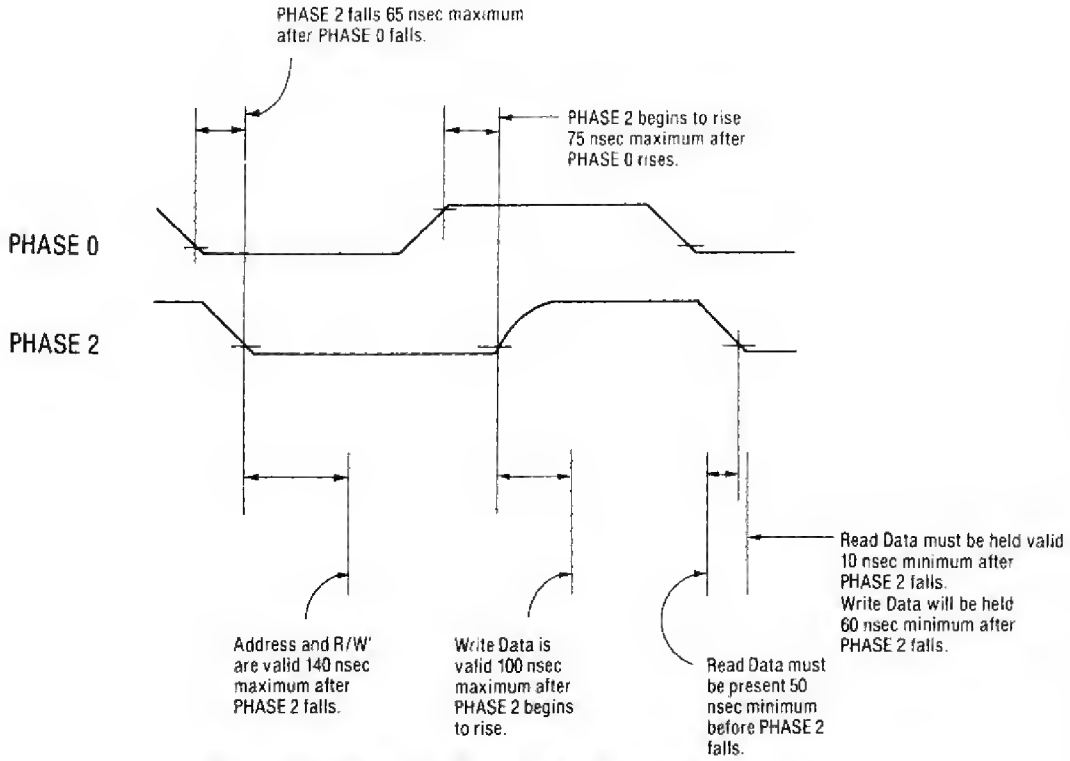


Figure 4.4 Some Worst Case 6502A Specifications.

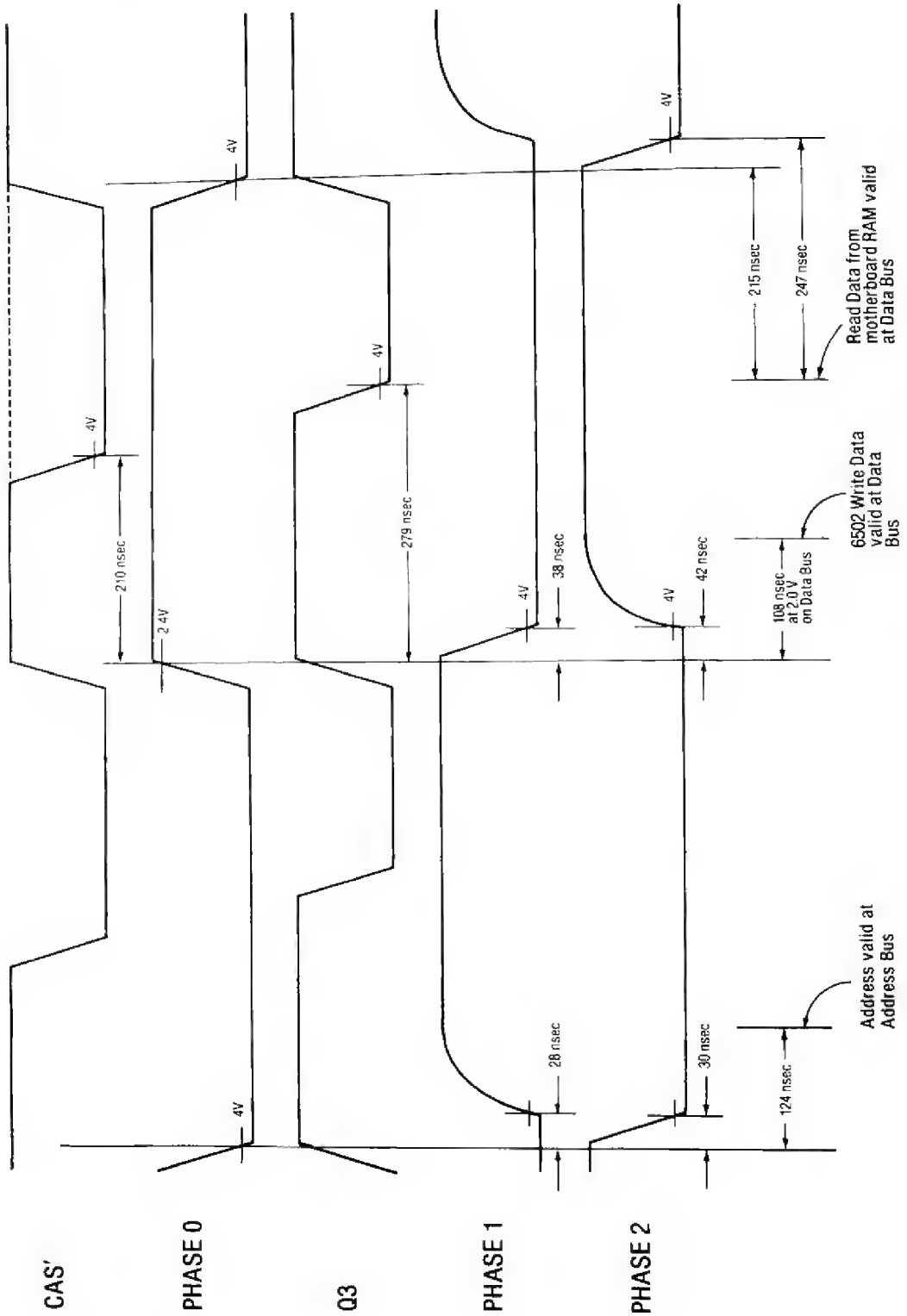


Figure 4.5 Experimental 6502A Timing Relationships.

motherboard RAM. CAS' in the Apple falls 209.5 nanoseconds after PHASE 0 rises, and the 6502A sets up write data well before this. The write data valid before CAS' criterion would not be met by a 1 MHz 6502 exhibiting worst case timing characteristics. Incidentally, the worst case 1 MHz 6502 would not meet this test in the old Apple II either, but the timing would be about 20 nanoseconds less critical.

4. Read data from motherboard RAM becomes valid at the 6502 about 215 nanoseconds before PHASE 0 falls or about 247 nanoseconds before PHASE 2 falls. This varies with the RAM chips, and the computer under test had OKI Semiconductor 200-nsec RAM chips installed. You could use much slower RAM chips in the Apple IIe, and the read data would still be set up long before necessary for 6502 reading. Auxiliary card RAM data becomes valid about 80 nsec later, but this is still far ahead of PHASE 2 falling.

6502 performance creates guidelines for motherboard and peripheral slot devices which communicate with it. Some of these guidelines are listed below. Specific timing of 6502 communication with various Apple devices will be discussed in chapters covering those devices, so Chapters 5—9 should be studied to clarify the details of 6502 communication in the Apple.

1. The 6502 address can be read by a peripheral card before Q3 falling during PHASE 0 in time to trigger a DMA action that same cycle.
2. Write data from the 6502 can be clocked to a peripheral by the falling edge of PHASE 0.
3. Read data should be valid on the data bus by 50 nanoseconds before the end of PHASE 2, and should stay valid at least 10 nanoseconds after PHASE 2 has fallen. The minimum specified PHASE 0 falling to PHASE 2 falling delay is 5 nanoseconds, and the maximum delay of data through the 74LS245 peripheral slot data bus driver is 18 nanoseconds, so read data from peripheral cards should be valid 63 nanoseconds before PHASE 0 falls.

The requirements for read data being on the data bus before and after PHASE 2 can be met in a peripheral card by gating read data with **DEVICE SELECT'. This signal does not overlap PHASE 2, but the data stays valid on the peripheral data bus and main data bus until after PHASE 2 anyway.** When either of these buses is floated (when all devices on the bus present a high impedance to the bus), **the last valid data on the bus at the time it was floated remains valid until the bus is brought back under positive control.** Therefore, if the peripheral data bus is floated just before PHASE 2 falls, the last valid data before the bus was floated will still be propagated through the bidirectional bus driver, and the 6502 will still read the data correctly.*

APPLE PROGRAMMING

There are four levels at which programs can be written in the 6502 based Apple: 6502 machine language, 6502 assembly language, high level compiler language, and high level interpreter. The order of listing is from most difficult to least difficult.

A **6502 machine language program** is a series of numeric bytes. The bytes are stored in sequence in memory where the 6502 accesses them by incrementing the address bus and reading the program while executing. 6502 machine language instructions consist of one, two, or three bytes in succession. Each instruction consists of an **op code** and possibly a 1- or a 2-byte **operand**. Execution of a 3-byte instruction requires three cycles to fetch the instruction plus additional cycles to execute the instruction.

The 6502 has a set of **internal registers** which are manipulated by the program. A 6502 program performs its functions by overseeing the interplay among the internal registers and memory. The 6502 internal register complement is made up of five 8-bit registers and the 16-bit Program Counter. The following is a list of the registers and their functions:

REGISTER	FUNCTION
Program Counter	Contains current address of instruction being executed.
Accumulator	Principle arithmetic and logical register.
X-Register	Index register.
Y-Register	Index register.
Stack Pointer	Contains current stack address.
Status Register	Contains flags indicating 6502 operating modes and logical results of instructions.

*All of the tri-state buses in the Apple IIe hold the previous data valid for a long time when they are floated. There are several instances where this characteristic determines operational features, and some where it is necessary for correct operation of a device. Some of these instances are noted in the timing examples of following chapters.

Generally, programs center around the Accumulator and memory with the X- and Y-registers being used for address indexing. Values of the Program Counter, Stack Pointer, and Status Register are automatically kept by the 6502 and don't usually have to be accessed directly by the program. Provisions exist for direct control of the Stack Pointer and processor status. The Program Counter is controlled by the flow of the program.

The following is a 3-instruction 6502 machine language program listed in hexadecimal:

OP CODE	ADDRESS LOW	ADDRESS HIGH
AD	89	1D
85	16	
00		

The first instruction loads the 6502 Accumulator from address $1D89. The second instruction stores the 6502 Accumulator contents at address $16. The final instruction is a BREAK instruction which terminates the program. The purpose of the program is to transfer the contents of $1D89 to $16. Machine language programs may be entered and executed from the Apple monitor using methods described in the *Apple II Reference Manual for IIe Only*.

Assembly language is a way of writing machine language programs with computer assistance. Many aspects of machine language programming are performed better by computer than by humans. Some such aspects are remembering op codes, addition and subtraction of addresses, remembering addresses of subroutines, and checking for syntax errors. Assembly language assists the programmer with these and other details and allows the use of English language symbology for addresses, operands, and opcodes. A prime goal in computer language development is English language compatibility.

The same program that was listed above in machine language is listed here in assembly language.

```
LABEL   OP CODE ADDRESS   COMMENT

RESTORE   LDA     $1D89   RESTORE SAVED POINTER
          STA     $16
          BRK
```

This program contains English language which cannot be executed by the 6502. A computer can, however, take this program and convert it to a 6502 machine language program. A program that does this is an assembler. An assembler takes an assembly language **source program** and assembles from it a machine language **object program**.

6502 programs can be assembled in disk-based Apples using any of several commercially available assemblers. This is the best way for most Apple owners to write extensive 6502 programs. Compared to almost any computer, minicomputer, or microprocessor machine language, 6502 machine language is very simple to use. This extends to 6502 assembly language. There are only 56 mnemonic codes to learn, and the logical selection of mnemonics makes this a simple learning task.

The simple instruction set has advantages and disadvantages. The chief disadvantage is that in some instances a program will require more instructions to accomplish a purpose than it would if powerful special purpose instructions were available. This can result in loss of speed and waste of memory space in some programs. One should not get the idea that the 6502 is without powerful features. It has a very versatile set of addressing modes and a decimal mode which speeds execution of certain types of programs considerably. It's just that there are more powerful and complex microprocessors around.

Another way to produce machine code involves the use of **compilers**. Programs may be written in high level languages such as BASIC, Pascal, and FORTRAN. High level language programs consist of powerful symbolic commands such as "PRINT" and "=". A 6502 cannot execute such commands, but computer programs (compilers) can examine such commands and produce 6502 machine language code which will cause the 6502 to perform the indicated functions.

A compiler is like an assembler in that it takes a symbolic language source program and translates it to a machine language object program. It is different from an assembler in that whole machine language routines are generated by a single compiled instruction. Only one machine language instruction is generated by an assembly language instruction. High level languages are much more powerful than assembly language in easing the task of the programmer. However, machine language code compiled from high level languages by a compiler is generally less efficient than code assembled from assembly language programs. The programmer has direct control over the machine code generated in assembly language, and human minds generate more efficient code than compiler programs. With compilers, as with assemblers, symbolic source code

must be entered with the assistance of a text editor. The compiler source code must be compiled into machine language object code before a program can be run.

In some ways, this process of converting a high level language program to machine code is a nuisance. The object code must be compiled before it can be run and debugged. In an alternate process, a high level language program can be interpreted as it is run. The interpreting program examines the high level language commands during program execution, and it directs program flow to resident machine language routines which perform the indicated functions. This is the process used with the Applesoft and Integer BASIC languages supplied with the Apple IIe computer.

Both the compiling and interpreting processes are available for high level languages in the Apple. In addition to the Applesoft and Integer BASIC interpreters in common usage, compilers are available that will compile stored Applesoft and Integer programs into machine language routines. These routines will execute much faster than an interpreter performing the same function, because the time-consuming interpretation process is separated from execution. Compilers and interpreters for other high level languages are also available.

Which language should you program in—assembly language or a high level language? The answer depends not only on the programmer's background, experience, and personal preference, but also on the requirements of the particular application. Assembly language is fastest and provides the most efficient use of memory space. Some programs requiring speed or large amounts of memory can be written only in assembly language. Machine code compiled from high level language source code offers a great combination of programming ease and speed of execution. BASIC programs interpreted and executed by the firmware interpreter supplied with the Apple are the easiest of all to write and debug, but very slow in execution.

Whatever language you program in, the 6502 will be executing machine language code. All of the important Apple operating systems—BASIC, Pascal, DOS, ProDOS, the monitor, and the Mini-Assembler—are machine language code which was originally written in assembly language.

An important footnote while discussing Apple programming languages and operating systems is the secondary MPU which may replace the 6502 via the DMA′ line or co-process with the 6502 from the auxiliary slot. These secondary MPUs greatly expand the possibilities of what one might find operating in the Apple. Of particular importance is the Z80 card and the associated CP/M operating system. CP/M (Control Program for Microprocessors) is a disk operating system developed by Digital Research company for which many programs are available. The Apple with Z80 card is potentially the most important CP/M computer.

DMA IN THE APPLE

DMA (Direct Memory Access) refers to a form of fast I/O in which the I/O device directly accesses memory. In DMA, the MPU is removed from the data transfer path between the device and RAM. There is no program sequence loading data from the source and storing it at the destination.

The video scanner access to RAM while PHASE 0 is low is a form of DMA referred to as simultaneous DMA. It is possible because RAM can be accessed twice as fast as the MPU access in the Apple, and because actual MPU data transfers occur only during a short period at the end of the 6502 machine cycle. This simultaneous DMA is completely transparent to the MPU. It has no effect on program execution since it does not affect the 6502 machine cycle.

A second form of DMA is cycle stealing. In cycle stealing DMA, the clock input to the MPU is stopped for a machine cycle, and the DMA device accesses RAM while the MPU is stopped. Thus, a cycle is stolen from the MPU. This type of DMA slows program execution.

Cycle stealing DMA is implemented in the Apple IIe. The DMA′ line is wired to pin 22 of the peripheral slots, and any peripheral card can directly access RAM by pulling DMA′ low while PHASE 0 is low and holding DMA′ low until PHASE 0 goes high then low again. Pulling DMA′ low forces the MPU address bus driver to a high impedance state and gates off the PHASE 0 clock input to the MPU. With DMA′ low, even though PHASE 0 goes high at pin 40 of the peripheral slots and everywhere else on the motherboard, PHASE 0 does not go high at pin 37 of the MPU. The 6502 waits with PHASE 1 high, PHASE 2 low, and inward direction of the MPU data bus connection. The MPU is thus isolated from the address bus and data bus, and the peripheral card can take control of both buses and the R/W′ line. DMA devices should communicate with the data bus at the end of PHASE 0 as the 6502 does. In

the Apple, PHASE 1 belongs to the video scanner, and devices do not respond to addresses except during PHASE 0.

Figure 4.6 shows the timing for stealing a cycle from the 6502 in the Apple IIe. The DMA device access is similar to 6502 access. The address bus should contain a valid address before RAS' rising during PHASE 1 so the MMU will have time to respond with control of the multiplexed RAM address bus and CXXX line before RAS' falls.* Write data to RAM should be set up before CAS' falls, and read data can be clocked to the DMA device by PHASE 0 falling.

It would be wise not to steal too many cycles at a time, because if the clock is stopped too long, the 6502 will lose its internal data, and the program will crash. It is not clear how long the clock can be stopped before the 6502 operation becomes unreliable. The MOS Technology data sheet lists the maximum PHASE 0 pulse width at 520 nanoseconds. This is clearly not accurate because every Apple in the world operates very well with a 629-nanosecond PHASE 0 pulse on one out of 65 cycles. The *Osborne 4 & 8-Bit Microprocessor Handbook* (copyright 1981, McGraw-Hill, Inc. by Adam Osborne and Gerry Kane) states that you cannot stretch the PHASE 1 or PHASE 2 clock on MCS6500 microprocessors. Osborne and Kane must have read the same data sheet. The Synertek data sheet for SY650X microprocessors shows a maximum cycle time of 40 microseconds. This seems to indicate that you can perform DMA in the Apple for 40 consecutive PHASE 0 cycles without adversely affecting the Apple. The Rockwell International data sheet shows a maximum cycle time of 10 microseconds which is probably a good number.

It happens that Steve Wozniak, the original designer of the Apple II, knows a great deal about this subject. In a conversation with the author, Mr. Wozniak revealed that his first design for the Apple II used a different method of scanning memory for video output than the simultaneous DMA used in his final design. When he was designing the Apple II, RAM chips which could be accessed at 2 MHz were just becoming available. As a consequence, the early design had a 1 MHz 6502 from which 40 out of 65 cycles were stolen for memory scanning. The 6502, therefore, effectively ran at about 385 KHz (25/65 x 1 MHz). What Mr. Wozniak found out was that you could hold off the clock on a new 6502 for 40 microseconds, but that as the chip cooked in, this hold-off capability deteriorated. He found it necessary to keep new 6502s handy so he could replace the MPU when the Apple started to malfunction. The 6502s were not failing. They were just becoming unable to retain data for 40 microseconds with the clock stopped. Mr. Wozniak speculates that the reason for this is a deterioriation in capacitance of internal elements after the 6502 is run for a while.

Mr. Wozniak never determined the maximum reliable hold-off time of the 6502 experimentally. The availability of faster RAM chips enabled him to design the superior version of the Apple II which was eventually released. His feeling is that it is safe to hold off the clock to a 6502 for five microseconds, which is the value used in Microsoft's Z80 card for the Apple II.* He also cautions that any experimental determination of this capability would have to be performed on new 6502s, used 6502s, and very used 6502s.

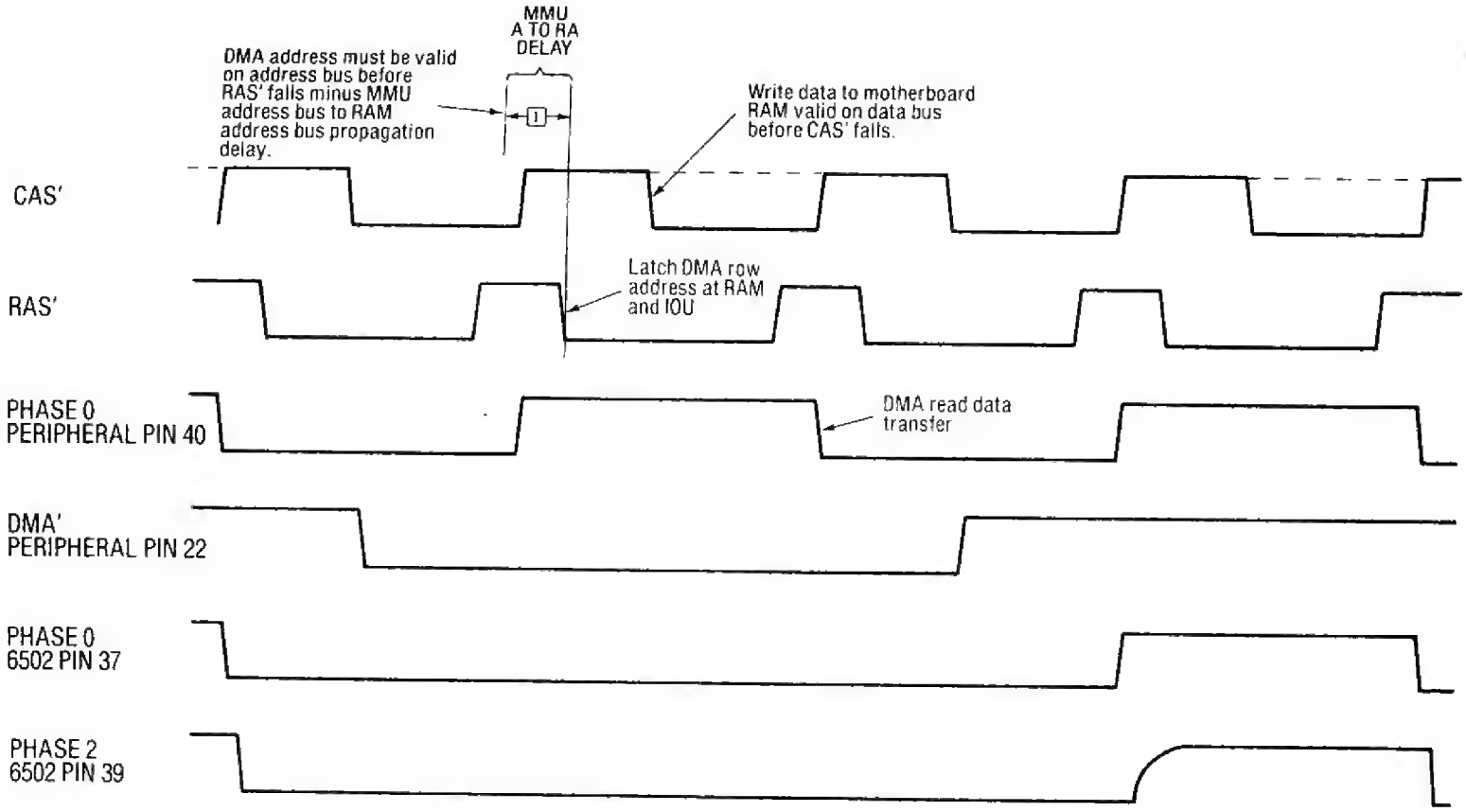
It's pretty obvious that the DMA' line can be used for more than just direct access to RAM. Since 6502 control of the Apple is via address decode, any device controlling the address bus can control the Apple. For example, a very simple peripheral card could change Apple screen modes via pushbutton. It would just have to steal a single cycle from the 6502, and gate $C05X to the address bus during PHASE 0 to select a screen mode depending on which button had been pressed. The most common use of the DMA' line in the Apple is to operate an MPU other than the 6502 from a peripheral slot. A Zilog Z80 card, Motorola MC6809 card, Intel 8088, or what have you can be plugged in to allow control of the Apple by the owner's favorite MPU. These cards gain access to the Apple via the DMA' line.

The DMA' line has no effect on video scanner access to RAM since the video scanner is isolated from the address bus. In other words, the scanner access to RAM is transparent to the DMA device, just as it is transparent to the motherboard MPU.

The 6502 designers intended that the **READY line** be used for DMA. Their idea was to stop the 6502 in a read cycle, and bring an external tri-state address bus driver to high impedance with the READY line while DMA took place. The READY line in the Apple has no effect on the tri-state

*There is no published specification for MMU address bus to multiplexed RAM address bus and CXXX propagation delay. I believe, but I don't guarantee, that DMA peripherals will work if they control the address bus before RAS' rising during PHASE 1. See Chapter 5 for more information on MMU signals.

*The Apple IIe version of Microsoft's Z80 card does not perform DMA, but is a separate microcomputer with 64K of RAM which resides in the auxiliary slot and processes simultaneously with the 6502 on the motherboard. The Z80 cannot access motherboard circuits, but both the 6502 and the Z80 can access auxiliary card RAM.



1 Measured at 82 nsec in author's Apple IIe. Maximum MMU address bus to RAM address bus propagation delay not published by Apple.

Figure 4.6 Cycle Stealing DMA.

address bus driver, so DMA can only be accomplished by pulling the DMA' line low. The DMA' line can be pulled low in conjunction with the READY line, but after a number of cycles the 6502 will lose its internal data, because it has no input clock. This situation can be changed by soldering jumper X4 and cutting jumper X5 on the motherboard. With these jumpers reconfigured, pulling DMA' low will not prevent the 6502 PHASE 0 clock from rising and falling. It is therefore possible to design peripheral cards that perform DMA indefinitely with the 6502 in its wait state. When the DMA operation is complete, the 6502 is able to resume operation as if nothing had happened.

There is a **priority system** of DMA operation in the Apple in which the lowest peripheral slot has priority if more than one peripheral tries to perform DMA at the same time. The priority system is implemented by a DMA in/DMA out priority chain which goes from slot to slot. Pin 27 is the DMA input on each peripheral slot which tells a card that no higher priority card is performing DMA. Pin 24 is the DMA output by which DMA from lower priority cards is disabled. Slot 1 has the highest priority and Slot 7 has the lowest priority. Pin 24 on each slot is tied to pin 27 on the next slot with Slot 1, pin 27 and Slot 7, pin 24 not connected as shown in Figure 7.6.

In the priority system, when pin 27 is low, a card should not attempt DMA because a higher priority card is performing DMA. The card should also bring pin 24 low so lower priority DMA cards are disabled. If pin 27 is high, a card may perform DMA. It should bring pin 24 low while performing DMA, and bring it high while not performing DMA. Non-DMA cards are always designed with pin 24 jumpered to pin 27 so they can be inserted between DMA cards in the peripheral slots. This keeps the priority chain intact. There can be no empty slots between DMA cards in a priority chain.

The DMA priority chain can be used to prioritize other functions besides DMA. Apple did this with its 12K firmware cards which substitute peripheral card ROM for motherboard ROM. Several firmware cards can be placed in a priority chain which prevents ROM on two separate cards from being simultaneously enabled. If two groups of cards use the DMA chain for different purposes, they may have to be separated by an empty slot or by a card with pin 27 or pin 24 open. For example, a firmware card in Slot 4 would interfere with the operation of a DMA card in Slot 5. Even when a firmware card is enabled, cycles are available when RAM or I/O is accessed in which the DMA priority line stays high. A DMA device down line from the firmware card will operate if it needs only to steal an occasional cycle and can wait for the firmware card to access RAM.

6502 INTERRUPTS IN THE APPLE IIe

There are actually four types of 6502 interrupt: RESET', NMI', IRQ', and the BREAK instruction. Each has its own unique characteristics and purposes as determined by the 6502 design. The hardware interrupts are connected to the peripheral slots, and RESET' is also connected to the RESET key and to pin 15 of the IOU. The BREAK instruction is a **software interrupt.** The response of the 6502 to interrupts in the Apple is determined by programs contained in the E0—FF ROM.

RESET'

Except for RESET', the general idea of the interrupts is to interrupt the MPU, perform an interrupt handling routine, and then return to the interrupted program. The general idea of RESET' is to interrupt the MPU and go to a coherent program start. There are no provisions in the 6502 response to a RESET' for saving internal registers and returning to the place where the program was interrupted. The 6502 response to RESET' is as follows:

1. Pull three meaningless values from the stack.
2. Fetch the RESET routine address from $FFFC and $FFFD, low byte first.
3. Set the IRQ' disable bit of Status Register; leave other Status bits as they are.
4. Begin execution of RESET routine.

The reason for the three meaningless stack accesses is that RESET' is a modified form of the other interrupts with R/W' forced high. Accordingly, the Stack Pointer is decremented while the three values are being read from memory, as if data were being pushed to the stack. Normally, the Stack Pointer is incremented during pull operations and decremented during push operations.

The RESET' sequence creates a "fingerprint" on the address bus which the MMU uses to detect a system reset, even though RESET' is not connected to the MMU. Any time the address bus contains three Page 1 addresses in sequence followed by $FFFC, the MMU assumes that a system reset is occurring, and resets all of its soft switches. Unfortunately, the MMU does not distinguish between ascending Page 1 references and descending Page 1 references, so Page 1 resident programs which read

$FFFC or vector to the contents of $FFFC/$FFFD via "JMP ($FFFC)" inadvertently reset the MMU soft switches.*

One of the actions taken by the MMU when it detects a system reset is to disable high RAM for reading. For this reason, the 6502 action following the reset sequence is determined by the contents of the E0—FF ROM. The contents of $FFFC/$FFFD in this ROM are $FA62, the address of the reset handler. This handler performs a number of housekeeping functions such as initializing the video display mode. It also performs several operations that make the Apple IIe reset fairly unique among microcomputers.

One unique feature of the Apple reset is that the reset handler eventually passes program control to a RAM vector. This means that the ultimate response of the Apple's RESET key is controllable by software. At power up, the RAM RESET vector ($3F2 and $3F3) is set, and from that point, it may be changed to any 6502 address by whatever program is controlling the Apple at a given moment. If the Apple has no disk drive, the RAM RESET vector is set at power up to enter BASIC. If there is a disk drive, the Apple enters the bootstrap routine contained in ROM on the disk controller. The RAM RESET vector is usually set by software loaded from the disk.

The automatic startup of the disk at power up is a feature that was added to the Apple II monitor when disks became popular. The new monitor was called the Autostart Monitor, and the Apple IIe reset handler is the Autostart Monitor with additions that accommodate Apple IIe features like the open Apple and close Apple keys and the 80-column display.

Autostart firmware only boots the disk on a reset that occurs at power up. Other resets cause program flow to go to the address contained in the RAM RESET vector. The firmware uses a code at $3F4 to determine whether a given RESET' was initiated at power up or not. The code is never properly set at power up, but the power-up reset sets the code so the following resets will not be "cold starts." The power-up byte ($3F4) must be the exclusive-OR between $A5 and the contents of $3F3, or a power-up reset will be performed when RESET is pressed. Any program can scramble the power-up code and force a "cold start" when RESET is pressed.

If the open Apple or close Apple key is being held down when a reset occurs, special versions of the reset handler are performed. A close Apple (right Apple) reset forces performance of the firmware diagnostic routines. After the diagnostics, all of RAM is blanked, and the following reset will be a power-up reset. An open Apple (left Apple) reset causes meaningless values to be stored in two locations of every memory page from Page $01 through Page $BF before the power-up byte is checked. The power-byte is among those locations modified, so the power-up reset is performed and the disk is booted. The open Apple reset thus overides software control of the power-up byte, and forces a disk boot any time the operator desires it.

It is obvious that it is not necessary to modify two bytes in 191 memory pages to fix the power-up byte for a forced disk boot. It should be equally obvious that this is Apple's way of protecting commercial programs from examination by you, the owner of the Apple IIe. This particular protection scheme is only formidable enough to protect data from the most casual attempts at observation, and Apple realizes that. But by clobbering a little data, Apple avoided pulling the rug out on the software suppliers who had fixed the power-up byte in their Apple II programs so users couldn't reset their computers.

NMI' and IRQ'

The NMI' and IRQ' lines are both connected only to the peripheral slots in the Apple. The IRQ' is the normal I/O interrupt signal because it can be enabled or disabled under program control. The idea of a non-maskable interrupt is to take action which has higher priority than any programming purpose. For example, an Apple may be required to take emergency action in the event of a failure in a manufacturing robot that it is controlling. The non-maskable interrupt can also be used in monitoring the Apple operation from a remote panel or single instruction step execution of 6502 programs. These applications would, of course, require peripheral card designs.

The interrupt sequence is similar for either NMI' or IRQ'. The 6502 first completes execution of the current instruction. Then the following sequence occurs in the case of NMI' or IRQ', with interrupt requests enabled:

1. Program Counter is pushed on stack, high byte first.
2. Processor Status is pushed on stack with BREAK bit reset.
3. Contents of interrupt vector (NMI' = $FFFA—$FFFB; IRQ' = $FFFE—$FFFF) are fetched, low byte first.

*Please see Table 4.1 near the end of this chapter for details of the 6502 instructions.

4. Interrupt routine is begun with interrupt requests disabled.

There is a basic hardware difference between NMI' and IRQ' in the 6502. NMI' is **edge sensitive** like a clockpulse input, and IRQ' is **level sensitive.** A typical order of events with NMI' is:

1. The NMI' line drops low.
2. The NMI handling routine is executed with interrupt requests disabled.
3. The NMI' line is brought high.
4. Normal program flow is resumed with interrupt requests enabled or disabled as they were before the non-maskable interrupt occurred.

A second non-maskable interrupt will not interrupt the routine of the first as long as the NMI' line is held low. Thus, while NMI' is not maskable by program control, it is hardware maskable in the sense that any interrupting device can prevent further interrupting by holding NMI' low. Recall that part of the NMI sequence is the disabling of interrupt requests, so the IRQ' cannot interrupt an NMI' handler unless the handler enables it.

A typical order of events with IRQ' is:

1. The IRQ' drops low.
2. The interrupt routine execution is begun with interrupt requests disabled.
3. The interrupt is acknowledged and IRQ' goes high.
4. The interrupt routine execution is completed and normal program flow is resumed with interrupt requests enabled.

Interrupt requests are disabled by the IRQ' sequence just as they are in the NMI' sequence. This prevents the still low IRQ' from immediately generating a second interrupt. The program maskable IRQ' can be used in any variety of implementation methods. The program must acknowledge and enable interrupts in a manner consistent with the protocol of the interrupting hardware. The point with IRQ' is to acknowledge the interrupt before enabling further interrupts, so that multiple interrupts are not generated inadvertently. Interrupt acknowledgements in the Apple usually consist of an access to one of the peripheral slot assigned addresses.

The enabling and disabling of IRQ' can be done fairly effortlessly in many applications. Either NMI' or IRQ' saves the Program Counter and processor Status Register on the stack before vectoring to the **interrupt handler**. The Status is saved before the interrupt disable bit of the Status Register is set. If, at the end of the interrupt handler, an RTI (ReTurn from Interrupt) instruction is executed, the Program Counter and Status Register are restored. Along with the rest of the Status Register, the pre-interrupt state of the interrupt disable bit is restored. Further interrupts are automatically disabled by the interrupt sequence, and the disable/enable status is automatically restored by the RTI instruction. The other 6502 registers (Accumulator, X-register, Y-register, and Stack Pointer) are not automatically saved by interrupts. These must be saved and restored by the interrupt handler if the application demands it.

In some applications it would be desirable to enable interrupt handlers to be interrupted. This sort of processing is handled well by the stack architecture. Return link information for each interrupt is simply stacked over each other, possibly several interrupts deep. All of the interrupts are eventually fully serviced when the congestion is reduced.

Any peripheral card may interrupt the 6502 in the Apple. If there is a possibility of multiple interrupt sources, the 6502 needs to be able to distinguish among the interrupting devices. This can be done by polling. In polling, the interrupt handler checks each peripheral slot to see if it caused the interrupt. Each card in a polling system must be capable of responding to an address prompt by placing its interrupt status on the data bus (normally MD7 of the data bus).

The peripheral slots have an **interrupt priority chain** which works exactly like the DMA priority chain. Card designs supporting the priority chain follow the same protocol as described in the section on DMA. As in other priority operations, Slot 1 has the highest priority and Slot 7 has the lowest priority. Cards in a priority chain control interrupts at lower priority cards and are controlled by higher priority cards. The priority chain does not eliminate the need for polling in a multiple interrupt source environment. Nor is the priority chain necessary to determine priority since this is determined implicitly by the order in which the interrupt handler polls the devices. Still, there are many conceivable uses for the priority chain. For example, a card may perform operations which will not tolerate interrupts from lower priority devices, but will tolerate interrupts from higher priority devices. Through the priority chain, system designs can be implemented to selectively enable high priority interrupts only.

There is a way in the Apple to determine priority of interrupts without any loss of time. This way is to have the interrupting card contain its own IRQ vector. In the Apple, any peripheral card can disable

motherboard ROM and steal ROM addresses. The interrupting card would only have to steal $FFFE and $FFFF to vector the Apple to its handler. This system could use the interrupt priority chain to prevent two cards from simultaneously responding to $FFFE or $FFFF.

The firmware implementation of NMI' and IRQ' handlers is very simple. An NMI' vectors straight to $3FB, where a JUMP instruction to the software NMI' handler must be stored. The IRQ' is handled differently. A short firmware routine that begins at $FA40 is executed. This routines first determines whether a BREAK instruction or an interrupt request is being processed. Both IRQ' and the BREAK instruction use $FFFE and $FFFF as their vector, and the IRQ' handler must distinguish between BREAK and an external interrupt request by checking the status that was pushed to the stack when the BREAK or IRQ' occurred. When it is determined that an external interrupt occurred, the program vectors to the contents of $3FE and $3FF. $3FE and $3FF should contain the address of the software IRQ' handler.

In distinguishing between BREAK and IRQ', the Apple firmware saves the contents of the 6502 Accumulator at memory location $45 and then modifies the Accumulator. The interrupted accumulator value must be retrieved from $45 if it is required for processing or restoration. Stacked interrupt applications requiring the saving of 6502 registers should save them on the stack. The accumulator value must be retrieved from $45 before pushing to the stack in the Apple.

The fact that memory location $45 is modified by the interrupt handler means that software to which $45 is critical cannot operate with IRQ' enabled and IRQ' based peripherals installed. This would seem to dictate that interruptable programs shouldn't store important information at $45 or call monitor subroutines that save the accumulator at $45 when accumulator contents are critical. This basic rule was ignored by Apple when it developed DOS 3.2 and 3.3, and it is possible for IRQ' based hardware to disrupt DOS and even to cause binary files to be stored on the disk using data from the wrong memory area.*

The very damaging consequence of the conflict over location $45 is that much software cannot operate with interrupting peripheral cards. This situation can be avoided if the software operates with high RAM enabled for reading with a custom interrupt vector and handler resident in high RAM. It can also be avoided if the interrupting peripheral disables motherboard ROM via INHIBIT' and substitutes its own firmware interrupt handler. But in all probability, the $45 problem will disappear as Apple's newly released enhancement to Apple IIe firmware gains acceptance. The interrupt handler in the enhanced firmware is far more extensive than that of the original Apple IIe firmware, and location $45 is not modified in the new handler. Please refer to **The Apple IIe Firmware Upgrade** in Chapter 6 for a general general description of the enhanced firmware. Refer to **The Enhanced Firmware IRQ'/BREAK Handler** later in this chapter for a description of IRQ' and BREAK handling with the new firmware.

*See "Go Ahead and Interrupt your Apple" by Dan Fischer and Morgan Caffrey, March and April '82 *SOFTALK*, for more information on the IRQ'/$45 problem.

The BREAK Instruction

The BREAK instruction is a software generated interrupt which is not disabled by the IRQ' disable bit of the Status Register. Its uses are not obvious, even to an experienced computer programmer who has not been exposed to it. Why would a program want to interrupt itself?

One use of BREAK is to make it the terminating instruction of 6502 programs rather than a RTS (ReTurn from Subroutine). The idea here is to have a program terminating routine which directs program flow to some sort of system utility. In this sense the BREAK is a programmable HALT instruction.

A second way of using BREAK is as a debugging breakpoint. When debugging or investigating software, it is often useful to stop a program at a specific address to examine program progress. The BREAK instruction is a very convenient way of doing this. Instead of overwriting three bytes of code with a JUMP instruction, only one byte is overwritten by the BREAK instruction. The program counter and processor status are saved on the stack as with IRQ' and NMI', so a BREAK handler can be written to insert break points and resume flow after investigation.

A third use of BREAK is to allow out-of-control 6502 programs to bomb gracefully. A misdirected program tends to lead program flow to an address where no program has been stored. But the MPU doesn't know there is no program there. The 6502 is like a dog in heat; it will try to execute anything it finds on the data bus. This can be chaotic in any system, but especially in a memory mapped I/O system like that of the the Apple. Printers or disk drives can start operating when random addresses are accessed by the MPU. It happens that, at power

up, much of RAM goes to a state of all zeroes. $00 is the op code of the BREAK instruction and, as a consequence, many bombed programs wind up executing a BREAK instruction. This is good, because the BREAK handler is usually designed to neatly terminate a program and enter a human communication utility. In this way, the BREAK instruction redirects the indiscriminant 6502. It is an interrupt upon crash instruction. The response of the 6502 to a BREAK instruction consists of the following sequence:

1. Program Count + 2 is pushed to the stack, high byte first.
2. Status Register is pushed to stack (BREAK bit set).
3. BREAK/IRQ' address is fetched from $FFFE and $FFFF, low byte first.
4. Program execution is begun at the address contained in $FFFE and $FFFF with interrupt requests disabled.

The difference between the external interrupt request and the BREAK command is the BREAK flag, which is shown in 6502 literature as bit 4 of the Status Register. The BREAK flag is conceptually different from the other status flags, however. It is not tested by any 6502 instructions, and there is no set or clear instruction for the BREAK flag. It can only be checked after the Status Register has been placed on the stack. It is checked by pulling the Status value from the stack and checking bit 4. **Rather than a bit of the Status Register, the BREAK flag seems to be a characteristic of the way processor Status is pushed to the stack.*** The BREAK flag exists only in RAM after a push to the stack operation in accordance with the following rules:

1. PHP command sets bit 4 in RAM (no significance).
2. Push Status resulting from NMI' resets bit 4 in RAM (no significance).
3. Push Status resulting from IRQ' resets bit 4 in RAM (identifies IRQ').
4. Push Status resulting from BRK command sets bit 4 in RAM (identifies BREAK interrupt).

BREAK status is meaningful only in an IRQ' handler. It can be checked in an IRQ' handler with the following sequence:

```
PLA
PHA
AND #%0001000
BNE BREAK.HANDLER
BEQ CONTINUE.IRQ
```

In Apple IIe firmware, BREAK processing is initially identical to IRQ' processing. But, after the interrupt is identified as a BREAK, the paths of program flow diverge. After the BREAK is detected, the interrupted Status is restored, possibly enabling interrupt requests. Then all interrupted 6502 register states are stored in $3A, $3B, and $45 through $49. At this point, the program flow vectors to the contents of $3F0 and $3F1.

The soft BREAK vector ($3F0 and $3F1) is loaded at power up with the address of a routine that displays the interrupted 6502 register states. Also, the instruction at interrupted Program Count + 2 is disassembled and displayed, and the system monitor is entered. This BREAK routine is adequate for terminating programs and inserting debugging breakpoints. Program status saved by the BREAK handler is available for restart of flow via the G (GO) command of the monitor. After power up, controlling software can set the soft BREAK vector to the address of a custom BREAK handler.

The Enhanced Firmware IRQ'/BREAK Handler

Until recently, Apple has not paid much attention to interrupt applications in the Apple II or Apple IIe. However, current Apple activity suggests that those neglectful days have passed. Much effort was made to make ProDOS fully support interrupting devices, and the Apple IIc was designed with several interrupting internal devices and a comprehensive firmware interrupt handler. Furthermore, Apple has developed a firmware upgrade to the Apple IIe which contains an Apple IIc compatible interrupt handler that doesn't modify location $45 but saves the accumulator and the rest of the free world on the stack. The upgrade is described generally in Chapter 6, but features of the IRQ'/BREAK handler are described here.* Some knowledge of the

*The above concept of the BREAK flag is based strictly on my own experiments. In no literature was I able to find a satisfactory description of specifically when the BREAK flag is set and reset. The concept of BREAK status being stored in bit 4 of the Status Register simply does not fit the way I found BREAK status to be stored and checked. Inside the 6502, there may well be a bit of the Status Register which keeps track of BREAK status. In any case, the BREAK status can only be checked by retrieving it from RAM after a push Status to the stack operation.

*Information here is based on a 9/7/84 preliminary version of the firmware upgrade. It is possible that some details will change in the final version.

memory management soft switches on the part of the reader is assumed in the following discussions. Operation of these soft switches is described in the **MEMORY MANAGEMENT** section of Chapter 5.

In the new Apple IIe firmware, the IRQ'/BREAK vector at $FFFE/$FFFF of the E0—FF ROM points to address $C3FA which is the starting address of the firmware interrupt handler. Assuming that high RAM is not enabled for reading, this means that interruptable software must operate either with INTCXROM set, or with SLOTC3ROM reset, or with a peripheral card with interrupt handler at address $C3FA installed in Slot 3. Thus, in general, **interruptable software must operate with the 80-column firmware enabled.** Note that if software is operating with high RAM enabled for reading, the contents of high RAM determine the features of IRQ'/BREAK handling without regard to the contents of motherboard ROM.

The new firmware interrupt handler is far more extensive than the old one. Its overall philosophy is to extend the 6502 response to IRQ' or BREAK before executing the software handler, and to extend the 6502 response to the RTI which terminates the IRQ' software handler. The 6502 IRQ'/RTI combination automatically handles stack storage (before software handler) and retrieval (after software handler) of its Program Counter and Status register. The new firmware handler extends this combination so that the pre-interrupt Accumulator, X-register, Y-register, Apple IIe memory configuration, and I/O STROBE' active peripheral slot ($Cn) are also stored and retrieved from the stack. Rather than simply containing an IRQ'/BREAK handler, the firmware contains an **IRQ'/BREAK handler** at $C3FA and a post-IRQ' **RTI handler** at $C3F4.

Initial processing is the same for both IRQ' and BREAK execution in the new firmware. The Accumulator, X-register, and Y-register are saved on the pre-interrupt stack, and the Apple IIe is set to a fixed memory configuration (INTCXROM set, all auxiliary card RAM disabled, and high RAM disabled for reading and writing). While the configuration is being set, the pre-interrupt configuration status is checked and saved in a **machine state byte** whose format is: D7—D0 equal the pre-interrupt states of ALTZP, 80STORE • PAGE2, RAMRD, RAMWRT, HRAMRD, HRAMRD • BANK1, HRAMRD • BANK2, INTCXROM. This machine state byte is stored at location $44 if a BREAK is being processed, and on the motherboard stack if an IRQ' is being processed. Programmers beware: if pre-interrupt high RAM is enabled for reading but disabled for writing, the post-IRQ' RTI handler enables high RAM for reading and writing!

In the initial IRQ'/BREAK processing, all auxiliary card RAM is disabled, including the ALTZP ranges ($0—$1FF and $D000—$FFFF). This creates the problem of losing access to data on the stack if ALTZP was set when the interrupt occurred. To solve this problem Apple has established the following **convention for coherent ALTZP switching**:

1. The auxiliary stack pointer is always set to $FF after first setting ALTZP.
2. The motherboard stack pointer is always saved at $100 of auxiliary RAM when setting ALTZP.
3. The auxiliary stack pointer is always saved at $101 of auxiliary RAM when resetting ALTZP.

This protocol is critical to interrupt processing because the software handler will have to go to the auxiliary card stack if it needs to access pre-interrupt stack data and ALTZP was set when the interrupt occurred. Programmers beware: interrupts must be disabled while switching ALTZP and saving the stack pointers at $100 and $101!

After initial interrupt processing, the paths of BREAK and IRQ' processing diverge. If an IRQ' is being processed, additional data is pushed to the stack, and the software IRQ' handler whose address is stored at $3FE/$3FF of motherboard RAM is entered. The states of the pre-interrupt (motherboard or auxiliary) stack and the motherboard stack at entry to the software IRQ' handler are as follows:

ON PRE-INTERRUPT STACK	ON MOTHERBOARD STACK
PCH	Machine State
PCL	Active Slot ($Cn)
6502 Status	$C3
Accumulator	$F4
Accumulator	
Accumulator	
X-register	
Y-register	

The active slot number is taken from $7F8 as part of **another convention.** Peripheral cards which respond to $C800—$CFFF addressing must place their **$Cn (n = slot number) identifying number at $7F8** when they are activated if their $C800—$CFFF firmware is to be interruptable. If the 80-column firmware is active, $7F8 contains $C3.

The bytes $C3 and $F4 are at the top of the motherboard stack at entry to the software IRQ' handler. Assuming that the handler does not disturb the

stack, a terminating RTI will result in execution of the program at $F4C3. This is the address of the IRQ' **RTI handler** in the new firmware. The RTI handler is the reverse of the IRQ' handler. It restores the interrupted I/O STROBE' active peripheral slot, memory configuration, Accumulator, X-register, and Y-register. Then it saves a "STA $C007" (set INTCXROM) or a "STA $C006" (reset INTCXROM) followed by an "RTI" on the stack below the interrupted 6502 status. Then this code is executed (via the RTS at $C4C0). This restores the pre-interrupt status of the INTCXROM soft switch and returns program flow to the interrupted program.

If a **BREAK** is being processed instead of an IRQ', then nothing is pushed to the stack beyond the 6502 registers. Instead, the machine state is stored at $44, and 6502 register values are retrieved from the stack. Then INTCXROM is reset (slot ROM enabled), and the 6502 register values are stored at locations $3A, $3B, and $45—$49 as they were with the old BREAK handler. Finally, a jump is made to the address specified at $3F0/$3F1 of motherboard RAM. If this is the address of the old firmware BREAK handler ($FA59), the register data is fetched from its zero page locations and displayed.

The big difference between this new BREAK handling firmware and the old firmware is that INTCXROM is reset and all auxiliary card RAM is disabled by the new firmware before entry to the $3F0/$3F1 specified handler. An unfortunate feature of the processing is that **if ALTZP is set when BREAK is executed, the firmware handler at $FA59 displays incorrect 6502 register values.** This is because the register values are stored on the auxiliary stack, then ALTZP is reset, then incorrect values are retrieved from the motherboard stack and displayed.

In summary, the new interrupt handler is Apple's way of making it easy for software publishers to support interrupts and of standardizing the way in which software publishers support interrupts. The approach seems a little heavy handed, but I sympathize with Apple's need to introduce standardization into software which comes from from so many different sources. I question the decision to have interrupts vector directly to $C3XX firmware since it requires that 80-column firmware be active when interrupts are enabled, and that the pre-interrupt I/O STROBE' active slot be saved and restored as part of IRQ'/RTI handling. An $FXXX resident interrupt handler could avoid vectoring to $C3XX. It could check INTCXROM, set INTCXROM, then jump to $C400—any I/O STROBE' active peripheral card that was thus deactivated would automatically be reactivated when INTCXROM was reset after interrupt handling.

I also question the need to reset ALTZP. While resetting ALTZP does make it easy to have an IRQ' handler resident in motherboard high RAM, negative consequences of resetting ALTZP include the necessity of a stack pointer saving protocol, the possibility of split motherboard/auxiliary RAM storage of interrupted critical values, and unreliable operation of the firmware BREAK handler. To say the least, if ALTZP is to be reset, the firmware BREAK handler should be rewritten to operate correctly if ALTZP was set at BREAK execution time.

Regardless of my minor objections, the enhanced interrupt handling firmware works, and it doesn't clobber location $45. Through it, Apple should achieve its goal of establishing a workable IRQ' protocol for the Apple IIe that is compatible with IRQ' protocol in the Apple IIc.

Priority Among Interrupts

There are priority considerations among the interrupts which determine what happens when more than one interrupt occurs at the same time. The general priority of interrupts is as follows:

Highest	RESET'
	NMI'
	BREAK
Lowest	IRQ'

In the event of simultaneous interrupts, RESET' overrides all other processor actions. If NMI' drops low while RESET' is low, the processor will not respond to it. Once the RESET' routine has been entered, however, the processor can be interrupted by NMI' or BREAK. For this reason, it may be best for a peripheral card to disable its NMI' generating circuitry when RESET' occurs and leave it disabled until signaled by the 6502 that the RESET' routine is accomplished. The idea of RESET' is to reset the whole sytem, not just the 6502. All interrupts set the disable interrupt flag of the Status Register as part of their initial sequence. This disables external interrupt requests only (IRQ').

If NMI' falls during IRQ' or BREAK execution (after the interrupt cycle is begun, but before the interrupt vector is fetched), then the NMI' vector is fetched instead of the IRQ'/BREAK vector. If an IRQ' cycle is thus aborted, then the NMI' is handled first and the IRQ' is handled later when interrupt request response is reenabled (assuming IRQ' is still low). If a BREAK cycle is thus aborted, then the

BREAK is never executed. The NMI' is handled, and when an RTI is executed, the 6502 program counter is set to the address of the aborted BREAK instruction plus two. This is a bug in the 6502 that is corrected in the 65C02 (the CMOS equivalent of the 6502 that is supplied with the Apple IIe firmware upgrade). When NMI' falls during BREAK execution in a 65C02, BREAK execution is first completed, then the NMI' execution cycle is performed.

In the event of simultaneous BREAK and IRQ' with IRQ' enabled, the processor would complete the BREAK instruction, fetching the contents of the IRQ'/BREAK vector and disabling interrupt requests. Then the IRQ'/BREAK handler would be executed with bit 4 of the top byte of the stack identifying the interrupt as a BREAK. In Apple firmware, the pre-BREAK Status is pulled from the stack as soon as a BREAK is identified (after some minor housekeeping in the enhanced firmware). This would enable interrupt requests in our example and allow the IRQ' sequence to begin, assuming IRQ' was still low. Following the RTI instruction at the end of the IRQ' handler, the BREAK routine would be reentered and its course would be run.

THE 65C02 MICROPROCESSOR

A recent development in the 6502 world has been the introduction of the 65C02 MPU. This MPU (manufactured by NCR, Rockwell, and alternate sources) is fabricated using CMOS technology, instead of the NMOS used in the 6502. The general advantage of CMOS over NMOS is lower power consumption, but the 65C02 also has some new instructions which make it operationally more powerful than its NMOS brother. A 65C02 can execute any 6502 program that doesn't depend on fine instruction execution timing, but a 6502 cannot execute 65C02 programs that utilize the new 65C02 instructions.

Apple uses the 65C02 MPU in the Apple IIc microcomputer, and they intend to convert the Apple IIe over to the 65C02. The plan is to retrofit older Apple IIe's with the 65C02 as part of the firmware upgrade package described in Chapter 6. This will maximize compatibility between the Apple IIe and the Apple IIc, and make it possible to write shorter and faster Apple IIe assembly language programs. Because the Apple IIe may become a 65C02 based computer in the future, some data on the 65C02 is given here and in other parts of *Understanding the Apple IIe*.

The 65C02 improvements consist of the addition of new instructions and addressing modes, and the removal of some old 6502 bugs. For the most part, differences between the 6502 and 65C02 are well documented in the partial NCR 65C02 data sheet in Appendix C at the back of this book. Descriptions here will therefore be limited to a few points whose ramifications are not made entirely clear by the data sheet. Please note also that details of 65C02 instruction execution are given in Tables 4.3 and 4.4 in an application note later in this chapter.

First, the NCR and Rockwell 65C02s are not identical. The Rockwell chip executes some instructions that are not part of the NCR 65C02 repertoire. These are the zero page instructions RMBn (Reset Memory Bit n) and SMBn (Set Memory Bit n), and the zero page relative branch instructions BBRn (Branch on Bit n Reset) and BBSn (Branch on Bit n Set). The op codes of these Rockwell instructions ($X7 and $XF) represent NOPs in the NCR chip. Apple appears to be using NCR compatible 65C02s in its computers, but the Rockwell chip works fine in the Apple IIe. Please refer to Tables 4.3 and 4.4 for details of the additional Rockwell instructions.

The READY line of a 6502 will not halt the MPU during a write cycle, but the 65C02 READY line will. This raises the question, "what happens to the Apple IIe data bus if READY is pulled low during a write cycle and is held low for a number of following write cycles?" If the 65C02 attempts to control the data bus constantly for a series of wait state write cycles, it will compete with motherboard RAM for control of the data bus near the end of PHASE 1. Investigation shows that this is not a problem. During a long series of wait state write cycles, the 65C02 controls the data bus only during that portion of the machine cycle in which it controls the data bus during a normal write cycle. Therefore, its data bus connection is at high impedance during the majority of PHASE 1 in all wait state write cycles, and motherboard RAM is free to control the data bus near the end of PHASE 1.

The fact that interrupts do not cause abortion of a BREAK instruction is listed as an operational enhancement of the 65C02 on page 3 of the data sheet. The data sheet is referring to non-maskable interrupts, not interrupt requests. In a 6502 or 65C02, IRQ' falling after a BREAK op code fetch does not interfere with BREAK execution. However, if NMI' falls after a BREAK op code fetch and before the interrupt vector is fetched in a 6502, then the NMI' interrupt vector is fetched, and the NMI' handler is executed. An RTI at the end of the NMI'

handler causes return to the address (plus two) of the BREAK instruction and probable program crashing. This bug is fixed in the 65C02. As the data sheet indicates, NMI' falling during BREAK execution results in NMI' execution after BREAK execution is complete.

The NCR data sheet refers to the new increment accumulator and decrement accumulator instructions as INA and DEA. I don't know why they do this, because these instructions are clearly just new addressing modes of the INC and DEC instructions. The new mnemonics should be INC A and DEC A or just INC and DEC as given in the Rockwell data sheet. The addition of the INC and DEC accumulator addressing modes means these instructions have all the addressing modes of the other 6502 read-modify-write instructions (ASL, LSR, ROL, and ROR).

Another notable feature of the 65C02 data sheet is the 5000-microsecond maximum cycle time in the AC characteristics table on page 3. I take this to mean that you can stop the clock for a guaranteed minimum of 5000 microseconds with PHASE 0 high, but not with PHASE 0 low. The Rockwell data sheet is more specific about the difference. It states: "The input clock can be held in the high state indefinitely; however, if the input clock is held in the low state longer than 5 microseconds, internal register and data status can be lost". The significance is that, when the Apple IIe DMA' line is held low, it forces the PHASE 0 input to the MPU to a low state. I therefore conclude that long term continuous DMA in the Apple IIe cannot be performed with a 65C02 any easier than it can be with a 6502. In either case, long term continuous DMA can only be performed by pulling DMA' low after the MPU has been stopped via READY low, and only after the X4 and X5 Apple IIe motherboard jumpers have been configured so the MPU clock is not stopped when DMA' is pulled low.

A feature of the 65C02 that does not show up in the NCR data sheet is that the new BIT immediate instruction operates differently than BIT in the other addressing modes. In the other addressing modes, BIT sets the negative, overflow, and zero flags based respectively on operand bit 7, operand bit 6, and the result of Accumulator • operand. The 65C02 BIT immediate instruction affects only the zero flag, not the negative and overflow flags.

A final point about 65C02 operation that I'd like to make is mildly speculative. The 65C02 is pin compatible with the 6502, and was designed as a direct but more powerful substitute for the 6502. To make it work in the Apple IIe, you simply remove the 6502 and plug in the 65C02. However, the 65C02 does not work reliably in the older Apple II. I believe that the reason for this is that the 65C02 (or at least an NCR 65C02) requires read data to be set up longer than a 6502 operating at the same frequency. RAM read data in the Apple II becomes valid at the MPU (about 60 nsec before PHASE 2 falls) much later than it does in the Apple IIe (about 250 nsec before PHASE 2 falls). Whereas the 6502 can handle the short RAM read data set up time, the 65C02 seems to have trouble with it.

I have performed limited experiments with 65C02s in an Apple II. Basically, I found that two NCR 65C02As (2 MHz?) and one NCR compatible GTE G65SC02P-2 (2 MHz) caused intermittent program crashing that got worse as the peripheral card data bus load was increased. The Rockwell R65C02P1 (1 MHz) that I tried caused no program crashes. The NCR 65C02 program crashes occurred only with certain data bus sequences. If an RTS instruction is preceded by a NOP or SBC, and the Apple II video data preceding the RTS opcode fetch is $A0, $A2, or $A9, then the carry flag is set during otherwise normal execution of the RTS instruction. This unwanted setting of the carry flag occurred as mentioned with all three NCR type chips. One of the chips also set the carry flag if the video data preceding RTS was $89, and the another one also set the carry flag if the video data preceding RTS was $89 or $E9. Note that $89, $A0, $A2, $A9, and $E9 are all immediate mode 65C02 instructions.

In these experiments, I did not conclusively prove that the problem with the 65C02 in the Apple II is short set up time of RAM read data. This is merely a highly educated guess upon which I would be willing to bet a paycheck (if only I had one). Setting the data up quicker definitely helps, because the bugs mentioned in the previous paragraph do not exist when the program resides in a 16K RAM card whose read data becomes valid just after Q3 falls during PHASE 0. In any case, I am suspicious of the validity of the NCR claim of 50-nsec minimum read data set up time in its 65C02.

SOFTWARE APPLICATION

6502/65C02 INSTRUCTION DETAILS

The state of the address bus and data bus on every cycle of operation are normally of no interest to the Apple programmer. However, there are non-obvious features of 6502 command execution which affect programming of I/O. This is a natural consequence of decoding I/O commands from the address bus. These address details are of particular interest to the assembly language programmer, but they affect some BASIC programs too.

Table 4.1 contains an example of every type of instruction sequence found in the 6502. It shows the state of the address bus and data bus for each cycle

Table 4.1 6502 Instructions.

	1	2	3	4	5	6	7	
1. DEX	$1000 $CA	$1001 IGNORE						ADDR BUS DATA BUS
2. ASL A	$1000 $0A	$1001 IGNORE						
3. PHA	$1000 $48	$1001 IGNORE	SPNT W DATA					
4. PLA	$1000 $68	$1001 IGNORE	SPNT IGNORE	SPNT+1 DATA				
5. RTS	$1000 $60	$1001 IGNORE	SPNT IGNORE	SPNT+1 PCL	SPNT+2 PCH	PCH,PCL IGNORE	PCH,PCL+1 NEXT OP	
6. RTI	$1000 $40	$1001 IGNORE	SPNT IGNORE	SPNT+1 STATUS	SPNT+2 PCL	SPNT+3 PCH	PCH,PCL NEXT OP	
7. BRK	$1000 $00	$1001 INGORE	SPNT W $10	SPNT-1 W $02	SPNT-2 W STATUS	$FFFE IRQLO	$FFFF IRQHI	
8. BEQ $10 (Z=0)	$1000 $F0	$1001 $10	$1002 NEXT OP					
9. BEQ $10 (Z=1)	$1000 $F0	$1001 $10	$1002 IGNORE	$1012 NEXT OP				
10. BEQ $F3 (Z=1) (PX)	$1000 $F0	$1001 $F3	$1002 IGNORE	$10F5 IGNORE	$FF5 NEXT OP			
11. LDA #$AA	$1000 $A9	$1001 $AA						
12. LDA $70 STA $70	$1000 $A5	$1001 $70	$0070 w DATA					
13. ASL $70	$1000 $06	$1001 $70	$0070 OLD DATA	$0070 W OLD DATA	$0070 W NEW DATA			
14. LDA $70,X STA $70,X	$1000 $B5	$1001 $70	$0070 IGNORE	$0090 w DATA				
15. ASL $70,X	$1000 $16	$1001 $70	$0070 IGNORE	$0090 OLD DATA	$0090 W OLD DATA	$0090 W NEW DATA		
16. LDA $5772 STA $5772	$1000 $AD	$1001 $72	$1002 $57	$5772 w DATA				
17. ASL $5772	$1000 $0E	$1001 $72	$1002 $57	$5772 OLD DATA	$5772 W OLD DATA	$5772 W NEW DATA		
18. JMP $5772	$1000 $4C	$1001 $72	$1002 $57	$5772 NEXT OP				
19. JSR $5772	$1000 $20	$1001 $72	SPNT IGNORE	SPNT W $10	SPNT-1 W $02	$1002 $57	$5772 NEXT OP	
20. LDA $5772,X (NO PX)	$1000 $BD	$1001 $72	$1002 $57	$5792 DATA				
21. LDA $57F2,X STA $57F2,X (PX)	$1000 $BD	$1001 $F2	$1002 $57	$5712 IGNORE	$5812 w DATA			
22. STA $5772,X (NO PX)	$1000 $9D	$1001 $72	$1002 $57	$5792 IGNORE	$5792 W DATA			
23. ASL $5772,X (NO PX)	$1000 $1E	$1001 $72	$1002 $57	$5792 OLD DATA	$5792 W OLD DATA	$5792 W OLD DATA	$5792 W NEW DATA	
24. ASL $57F2,X (PX)	$1000 $1E	$1001 $F2	$1002 $57	$5712 IGNORE	$5812 W OLD DATA	$5812 W OLD DATA	$5812 W NEW DATA	
25. LDA ($70,X) STA ($70,X)	$1000 $A1	$1001 $70	$0070 IGNORE	$0090 ADL	$0091 ADH	ADH,ADL w DATA		
26. LDA ($70),Y (NO PX)	$1000 $B1	$1001 $70	$0070 $72	$0071 $57	$5792 DATA			
27. LDA ($70),Y STA ($70),Y (PX)	$1000 $B1	$1001 $70	$0070 $F2	$0071 $57	$5712 IGNORE	$5812 w DATA		
28. STA ($70),Y (NO PX)	$1000 $91	$1001 $70	$0070 $72	$0071 $57	$5792 IGNORE	$5792 W DATA		
29. JMP ($5772) (NO PX)	$1000 $6C	$1001 $72	$1002 $57	$5772 PCL	$5773 PCH	PCH,PCL NEXT OP		
30. JMP ($57FF) (PX)	$1000 $6C	$1001 $FF	$1002 $57	$57FF PCL	$5700 PCH	PCH,PCL NEXT OP		

W - WRITE CYCLE
w - WRITE CYCLE IF STORING INSTRUCTION
PX - PAGE CROSSING
NEXT OP - OP CODE NEXT INSTRUCTION
X-REG = $20, Y-REG = $20
$70/$71 CONTAIN $5772 OR $57F2 AS NEEDED FOR ILLUSTRATION

Table 4.2 6502 Instruction Cross Reference.

	IMP	REL	IMM	ACC	ØPG	ØPG X	ØPG Y	ABS	ABS X	ABS Y	IND	IND X	IND Y
ADC AND CMP EOR LDA ORA SBC			11		12	14		16	2Ø 21	2Ø 21		25	26 27
ASL LSR ROL ROR				2	13	15		17	23 24				
BCC BCS BEQ BMI BNE BPL BVC BVS		8,9 1Ø											
CLC CLD CLI CLV DEX DEY INX INY NOP SEC SED SEI TAX TAY TSX TXA TXS TYA	1												
BIT					12			16					
BRK	7												
CPX CPY			11		12			16					
DEC INC					13	15		17	23 24				
JMP								18			29 3Ø		
JSR								19					
LDX			11		12		14	16		2Ø 21			
LDY			11		12	14		16	2Ø 21				
PHA PHP	3												
PLA PLP	4												
RTI	6												
RTS	5												
STA					12	14		16	21 22	21 22		25	27 28
STX					12		14	16					
STY					12	14		16					

of execution. LDA, DEX, ASL, PHA, and PLA were chosen to represent classes of instructions whose execution sequences are identical. Table 4.2 is keyed to Table 4.1. To find an example of any instruction and address mode, look up the instruction in Table 4.2, then see the referenced example in Table 4.1.

The op code of all instructions shown in Table 4.1 is assumed to reside at $1000. The X- and Y-registers both contain $20 in all examples. Y-indexed instructions are represented by X-indexed examples when Y-indexed execution is identical to X-indexed execution. When possible, LDA examples are used to represent storing instructions (STA, STX, STY), and in these examples the write cycles of storing instructions have a "w" following their address. Cycles that are always write cycles have a "W" following their address. The letters "PX" stand for Page Crossing. A few examples show the first

cycle of the next instruction. This is indicated by "NEXT OP" on the data bus.

At times, the 6502 addresses parts of memory which have nothing to do with a given instruction. This occurs when the 6502 is performing an internal operation in a cycle and really doesn't need to address anything. Indexing or branching across page boundaries always results in a superfluous access to an address in the wrong page.* It takes an extra cycle for the 6502 to increment or decrement the high portion of an address computed across a page boundary. A "LDA $5F72,X", for example, takes four cycles with no page crossing, and five cycles with a page crossing. STA instructions in which the possibility of a page crossing exists allow an extra cycle whether the page crossing occurs or not. The *Synertek Programming Manual* (May 1978) states that this is necessary to prevent a superfluous write to the wrong address.

There are other interesting points about 6502 addressing. The read-modify-write instructions (ASL, LSR, ROL, ROR, INC, DEC) always perform a double write to the valid address.* The first write cycle writes the same data that was read, and the second write stores the modified data. Pulling data from the stack results in a superfluous access to a wrong Page 1 address. All superfluous accesses to wrong addresses are on read cycles, and the resulting data is ignored by the 6502.

Example 30 of Table 4.1 illustrates an obscure 6502 bug; the JMP indirect instruction cannot fetch the new program counter value from two bytes in different memory pages. As shown in cycles 4 and 5 of example 30, a "JMP ($XXFF)" gets the next program counter state from $XXFF and $XX00, not from $XXFF and $(XX+1)00 as you would expect.* Because of this unexpected operation, Apple IIe programmers should not utilize "JMP ($XXFF)" unless their ultimate motive is to create confusion.

Three software applications of 6502 addressing details are in the controlling of the serial outputs, high RAM, and the disk controller. The speaker and cassette are toggle outputs which are usually made to toggle up and down at an audio rate. The speaker, for example, should not normally be accessed by instructions which make a double or quadruple access to $C030, because that would result in the speaker line toggling back and forth at 1 MHz. The idea is to toggle the speaker, wait a thousand microseconds or so, then toggle it again. Similar considerations exist for the C040 STROBE'. The programmer may select a single, double, triple, or quadruple strobe by utilizing one of the following instructions:

```
STA $C040           One strobe
STA $C040,X (X = 0) Two strobes
ASL $C040           Three strobes
ASL $C040,X (X = 0) Four strobes*
```

In BASIC, it helps to be aware of what machine language instruction actually performs the memory access when a PEEK or POKE instruction is executed. The following instructions perform the actual memory access in the Apple (where Y = 0):

```
Applesoft PEEK - $E76F:  LDA($50),Y
Applesoft POKE - $E781:  STA($50),Y
Integer PEEK   - $EEF9:  LDA($CE),Y
Integer POKE   - $EF0D:  STA($CE),Y
```

Correlating the PEEK and POKE instructions with examples 26 and 28 of Table 4.1 indicates that POKE instructions generate a double access to the POKE'd address, and PEEK instructions generate a single access to the PEEK'd address. For this reason, speaker or cassette control from BASIC should be performed by PEEK instructions: "A = PEEK(−16336)" or "A = PEEK(−16352)." As for the C040 STROBE', "A = PEEK(−16320)" generates a single strobe, and "POKE−16320,0" generates a double strobe.

The way that high RAM is controlled makes it a prime candidate for sneaky address bus manipulation. The operation of high RAM is covered fully in Chapter 5, but a small note about its operation belongs here. As described in Chapter 5, high RAM is configured for writing by two successive reads to $C081, $C083, $C089, or $C08B (see Table 5.5). For this purpose, one instruction can accomplish the same as two. "ASL $C081,X" with X = 0 performs the same task as "LDA $C081; LDA $C081". Read-modify-write, absolute indexed, no page crossing instructions generate two read accesses and two write accesses (one write access in a 65C02) to the computed address. This is more cute than valuable, but it does illustrate the potential of controlling peripherals by single instruction address sequences in the Apple.

A more important application of knowledge of addressing detail can be seen at addresses $B82A through $B842 of the DOS 3.3 RWTS subroutine. $B82A is the beginning of the WRITE DATA routine which writes coded data to a sector of the disk. Direction of disk operations is accomplished on the disk controller by a logic state sequencer, which is a

*Statements marked by an asterisk in this application note are true for the 6502 but not the 65C02.

programmed hardware controller. Simply put, writing data to the disk consists of syncing the writing loop of the logic state sequencer to the writing loop of the controlling software. The following program steps check for write protect, and reset the logic state sequencer to its idle location:

```
LDA $C08D,X     X = $60 IF SLOT 6.
LDA $C08E,X
BMI WPROTECT    BRANCH IF DISK WRITE
                PROTECTED
```

The program will fall through the branch if the disk is not write protected. From this induced idle state, the software can sync itself to the logic sequencer with the statement, "STA $C08F,X". This instruction performs a double access to $C0EF (assuming Slot 6). The first access is decoded in the disk controller to cause the logic state sequencer to leave its idle state and begin its write loop. The second access stores actual disk write data in the controller's input/output register. The controller will only accept data on the clockpulse after the one which started the logic state sequencer and on every fourth clockpulse afterward. The writing technique involves writing data in software loops that take exact multiples of four cycles to execute.

Persons wishing to imitate the writing technique of the RWTS subroutine should not substitute a "STA $C0EF" instruction for the "STA $C08F,X" at address $B83F of DOS 3.3. "STA $C0EF" will start up the software loop one clockpulse out of sync with the logic state sequencer, and the controller won't accept the write data. "STA $C0EF,X" will work with 0 in the X-register. The instruction must make a double access to $C0EF. Another address mode of instruction which will work is a STA (ZP),Y with no page crossing.

No doubt, the Apple controller's logic state sequencer was designed around the "STA $C080,X" instruction, since this makes it possible to have the disk in other slots besides Slot 6. Given the hardware, Apple disk programmers must understand addressing details to program the disk on this level.

As a reference for those who have a 65C02 installed in their Apple IIe, Tables 4.3 and 4.4 show the instruction execution details of the 65C02. These tables are nearly identical to Tables 4.1 and 4.2, but they are different to the extent that 65C02 instruction execution is different from 6502 instruction execution. 65C02 instructions and execution cycles that are different from 6502 instructions and execution cycles are printed in boldface in Tables 4.3 and 4.4.

Some of the features of 6502 instruction execution that were pointed out in the preceding paragraphs are not features of 65C02 instruction execution. Please note that in 65C02 instruction execution:

1. Indexing or branching across a page boundary results in a superfluous read access, but the superfluous access is to the program counter address rather than to the operand address plus or minus 256 (examples 10, 21, etc.).
2. Read-modify-write instructions result in only one write access to the operand address and a maximum of three read or write accesses to the operand address (examples 13, 15, etc.).
3. The "ASL $C040,X" example that is given above will result in only three consecutive strobes (example 23).
4. "JMP ($XXFF)" is performed correctly (example 30).

Table 4.3 65C02 Instructions.

	1	2	3	4	5	6	7	8
	ADDR BUS							
	DATA BUS							
1. OEK	$1000	$1001						
	$CA	IGNORE						
2. ASL A	$1000	$1001						
	$0A	IGNORE						
3. PHA	$1000	$1001	SPNT W					
	$48	IGNORE	DATA					
4. PLA	$1000	$1001	SPNT	SPNT+1				
	$68	IGNORE	IGNORE	DATA				
5. RTS	$1000	$1001	SPNT	SPNT+1	SPNT+2	PCH,PCL	PCH,PCL+1	
	$60	IGNORE	IGNDRE	PCL	PCH	IGNORE	NEXT OP	
6. RTI	$1000	$1001	SPNT	SPNT+1	SPNT+2	SPNT+3	PCH,PCL	
	$40	IGNORE	IGNORE	STATUS	PCL	PCH	NEXT OP	
7. BRK	$1000	$1001	SPNT W	SPNT-1 W	SPNT-2 W	$FFFE	$FFFF	
	$00	INGORE	$10	$02	STATUS	IROLO	IROHI	
8. BEQ $10	$1000	$1001	$1002					
(Z=0)	$F0	$10	NEXT OP					
9. BEQ $10	$1000	$1001	$1002	$1012				
(Z=1)	$F0	$10	IGNORE	NEXT OP				
10. BEQ $F3	$1000	$1001	$1002	$1002	$0FF5			
(Z=1) (PX)	$F0	$F3	IGNORE	IGNORE	NEXT OP			
11. LDA #$AA	$1000	$1001	$1002 D					
	$A9	$AA	IGNORE					
12. LDA $70	$1000	$1001	$0070 w	$1002 D				
STA $70	$A5	$70	DATA	IGNORE				
13. ASL $70	$1000	$1001	$0070	$0070	$0070 W			
	$06	$70	OLD DATA	OLD DATA	NEW DATA			
14. LDA $70,X	$1000	$1001	$1001	$0090 w	$1002 D			
STA $70,X	$B5	$70	$7F	DATA	IGNORE			
15. ASL $70,X	$1000	$1001	$1001	$0090	$0090	$0090 W		
	$16	$70	$7F	OLD DATA	OLD DATA	NEW DATA		
16. LDA $5772	$1000	$1001	$1002	$5772 w	$1003 D			
STA $5772	$AD	$72	$57	DATA	IGNORE			
17. ASL $5772	$1000	$1001	$1002	$5772	$5772	$5772 W		
	$0E	$72	$57	OLD DATA	OLD DATA	NEW DATA		
18. JMP $5772	$1000	$1001	$1002	$5772				
	$4C	$72	$57	NEXT OP				
19. JSR $5772	$1000	$1001	SPNT	SPNT W	SPNT-1 W	$1002	$5772	
	$20	$72	IGNORE	$10	$02	$57	NEXT OP	
20. LDA $5772,X	$1000	$1001	$1002	$5792	$1003 D			
(NO PX)	$BD	$72	$57	DATA	IGNORE			
21. LDA $57F2,X	$1000	$1001	$1002	$1002	$5812 w	$1003 D		
STA $57F2,X	$BD	$F2	$57	$57	DATA	IGNORE		
(PX)								
22. STA $5772,X	$1000	$1001	$1002	$5792	$5792 W			
(NO PX)	$9D	$72	$57	IGNORE	DATA			
23. ASL $5772,X	$1000	$1001	$1002	$5792	$5792	$5792 W		
(NO PX)	$1E	$72	$57	OLD DATA	OLD DATA	NEW DATA		
24. ASL $57F2,X	$1000	$1001	$1002	$1002	$5812	$5812	$5812 W	
(PX)	$1E	$F2	$57	$57	OLD DATA	OLD DATA	NEW DATA	
25. LDA ($70,X)	$1000	$1001	$1001	$0090	$0091	ADH,ADL w	$1002 D	
STA ($70,X)	$A1	$70	$7F	ADL	ADH	DATA	IGNORE	
26. LDA ($70),Y	$1000	$1001	$0070	$0071	$5792	$1002 D		
(NO PX)	$B1	$70	$72	$57	DATA	IGNORE		
27. LDA ($70),Y	$1000	$1001	$0070	$0071	$0071	$5812 w	$1002 D	
STA ($70),Y	$B1	$70	$F2	$57	$57	DATA	IGNORE	
(PX)								
28. STA ($70),Y	$1000	$1001	$0070	$0071	$0071	$5792 W		
(NO PX)	$91	$70	$72	$57	$57	DATA		
29. JMP ($5772)	$1000	$1001	$1002	$1002	$5772	$5773	PCH,PCL	
(NO PX)	$6C	$72	$57	$57	PCL	PCH	NEXT OP	
30. JMP ($57FF)	$1000	$1001	$1002	$1002	$57FF	$5000	PCH,PCL	
(PX)	$6C	$FF	$57	$57	PCL	PCH	NEXT OP	
31. LDA ($70)	$1000	$1001	$0070	$0071	$5772	$1002 D		
STA ($70)	$B2	$70	$72	$57	DATA	IGNORE		
32. JMP ($5772,X)	$1000	$1001	$1002	$1002	$5792	$5793	PCH,PCL	
	$7C	$72	$57	$57	PCL	PCH	NEXT OP	
33. BBS0 $70,$10*	$1000	$1001	$0070	$0070	$1002	$1003		
	$8F	$70	$72	$72	$10	NEXT OP		
34. BBS1 $70,$10*	$1000	$1001	$0070	$0070	$1002	$1003	$1013	
(NO PX)	$9F	$70	$72	$72	$10	IGNORE	NEXT OP	
35. BBS1 $70,$F3*	$1000	$1001	$0070	$0070	$1002	$1003	$1003	$0FF6
(PX)	$9F	$70	$72	$72	$F3	IGNORE	IGNORE	NEXT OP
36. $X3, $XB **	$1000							
$X7, $XF ***	$03							
37. $5C $5772 **	$1000	$1001	$1002	$FF72	$FFFF	$FFFF	$FFFF	$FFFF
	$5C	$72	$57	IGNORE	IGNORE	IGNORE	IGNORE	IGNORE

W - WRITE CYCLE
w - WRITE CYCLE IF STORING INSTRUCTION
D - ONE CYCLE EXTENSION OF ADC OR SBC IF DECIMAL MODE
P - PAGE CROSSING

NEXT OP - OP CODE NEXT INSTRUCTION
* - AVAILABLE IN ROCKWELL BUT NOT NCR 65C02
** - UNUSED OP CODES (NOPs) IN ALL 65C02
*** - UNUSED OP CODES (NOPs) IN NCR 65C02 ONLY

X-REG = $20, Y-REG = $20.
$70/$71 contain $5772 or $57F2 as needed for illustration.
Boldfaced type is used where 65C02 is different from 6502.

Table 4.4 65C02 Instruction Cross Reference.

	IMP	REL	IMM	ACC	ØPG	ØPG X	ØPG Y	ABS	ABS X	ABS Y	ABS IND	ØPG IND X	ØPG IND Y	ØPG IND	ABS IND X	ØPG REL	*
ADC AND CMP EOR LDA ORA SBC			11		12	14		16	20 21	20 21		25	26 27	31			
ASL LSR ROL ROR DEC INC				2	13	15		17	23 24								
BBRn BBSn **																**33 34 35**	
BCC BCS BEQ BMI BNE BPL BVC BVS		8,9 10															
BRA		**9 10**															
CLC CLD CLI CLV DEX DEY INX INY NOP SEC SED SEI TAX TAY TSX TXA TXS TYA	1																
BIT			11		12	14		16	20 21								
BRK	7																
CPX CPY			11		12			16									
JMP								18			29 30				32		
JSR								19									
LDX			11		12		14	16		20 21							
LDY			11		12	14		16	20 21								
PHA PHP **PHX PHY**	3																
PLA PLP **PLX PLY**	4																
RMBn SMBn **					13												
RTI	6																
RTS	5																
STA					12	14		16	21 22	21 22		25	27 28	31			
STX					12		14	16									
STY					12	14		16									
STZ					**12**	**14**		**16**	**21 22**								
TRB TSB					13			17									
02 22 42 62 82 C2 E2 ***			11														
X3 XB * X7 XF******																	**36**
44 ***					12												
54 D4 F4 ***						14											
5C ***																	**37**
DC FC ***								**16**									

* unused op codes $X3, $XB, $5C (NCR and Rockwell) and $X7, $XF (NCR) generate abnormal addressing modes.
** BBRn, BBSn, RMBn, and SMBn are found on Rockwell 65C02 but not NCR 65C02.
*** unused op codes for NCR and Rockwell 65C02.
**** unused op codes for NCR 65C02 but not Rockwell 65C02.

HARDWARE APPLICATION

D MANUAL CONTROLLER

How many times have you been working with your Apple and had to look up control addresses to select HIRES NO MIX or LORES MIX or any other screen mode? It's too bad, but screen mode selection isn't supported in the Apple firmware by escape codes or similarly easy interface. The Apple scheme of controlling operational features via soft switches is extremely effective for control by programs, but the operator at the keyboard is left without means of direct control unless the operating program supports it. This application note describes a simple DMA controller which allows the operator to override program control and manually select among the Apple features, including screen modes. I call this circuit D MAnual Controller. Not everybody likes dis name, but dat's not my problem!

Figure 4.7 is a schematic of D MAnual Controller. It works by stealing a single cycle from the 6502 and placing an address in the $C0XX range on the address bus. This action is initiated when the operator presses one of eight pushbuttons (or four momentary on-off-on switches). Six slide switches (or six DIP switches) configure the Controller so the pushbuttons will affect different Apple features—screen modes, annunciators, disk drives, memory configuration, etc. The concept is to place the operator switches on a small remote panel, connected by a 16-wire ribbon cable to the cycle stealing peripheral card. Figure 4.8 is a photo of an earlier prototype which controlled screen modes only.

D MAnual Controller can control some peripheral card functions as well as motherboard features. Those peripheral card functions which can be controlled are the ones normally programmed using DEVICE SELECT' addresses such as RAM card, firmware card, and disk controller management. Tables 4.5 and 4.6 are an operational summary of D MAnual Controller showing how some features are controlled. Some of these are only educational or cute, while others, like screen mode control and memory configuration, can be very useful. It is recommended that the configuration switches be left in the position in which you will most often need them, so you will have convenient manual control of the features which are important to you.

Even though Table 4.5 shows how to control Slot 6 disk drives using D MAnual Controller, it doesn't follow that disk I/O can be performed manually. D MAnual Controller is not capable of transferring data via the data bus. It can only turn the drives on and off, select between drives, configure the disk controller for different functions, and position the head. Please take note that turning a drive on and setting READ/WRITE to WRITE will clobber the data on a disk which is not write protected. It is suggested that you experiment with no disk or an unimportant disk in the drive. Manual control of the disk drive is educational, but its only practical function would be to assist in the development of advanced disk programs and formats, or to aid maintenance technicians and disk hardware developers. Incidentally, to step the head, turn the phases on and off sequentially while a drive is rotating. Stepping through the phases in ascending order moves the head toward track 34. Stepping in descending order moves the head toward track 0.

D MAnual Controller is based in hardware, and overrides program control. You can select features at any time, no matter what software or firmware is running. This can be very convenient for programmers while they are developing programs. The Controller does not lock out program control, though, so programs which repeatedly select a given mode will not appear to be affected when the Controller deselects that mode. In the Apple IIe, there is no way to lock out program control of the $C0XX control functions.

Circuit Operation

The heart of D MAnual Controller is a 74LS148 priority encoder which detects a button push and converts it to a 3-bit address. This address is latched in a 74LS374 when a button is pressed and placed on A2, A1, and A0 of the address bus at the first opportunity. The state of A7—A3 of the address bus and R/W' during the DMA cycle are determined directly by the six configuration switches. A15—A8 are always set to 11000000 during the DMA cycle, yielding an address in the $C0XX range, the critical control range of the Apple.

Pressing any of the pushbuttons causes the signal at pin 14 of the LS148 to go low. This signal is debounced and inverted and sent to a 74LS195 shift register for single cycle generation. If the DMA priority input is low, the shift register will shift the button press signal through, and a 1-cycle negative signal will be felt at pin 2 of a 74LS74 flip-flop. The LS195 is clocked by PHASE 1 rising, so this 1-cycle

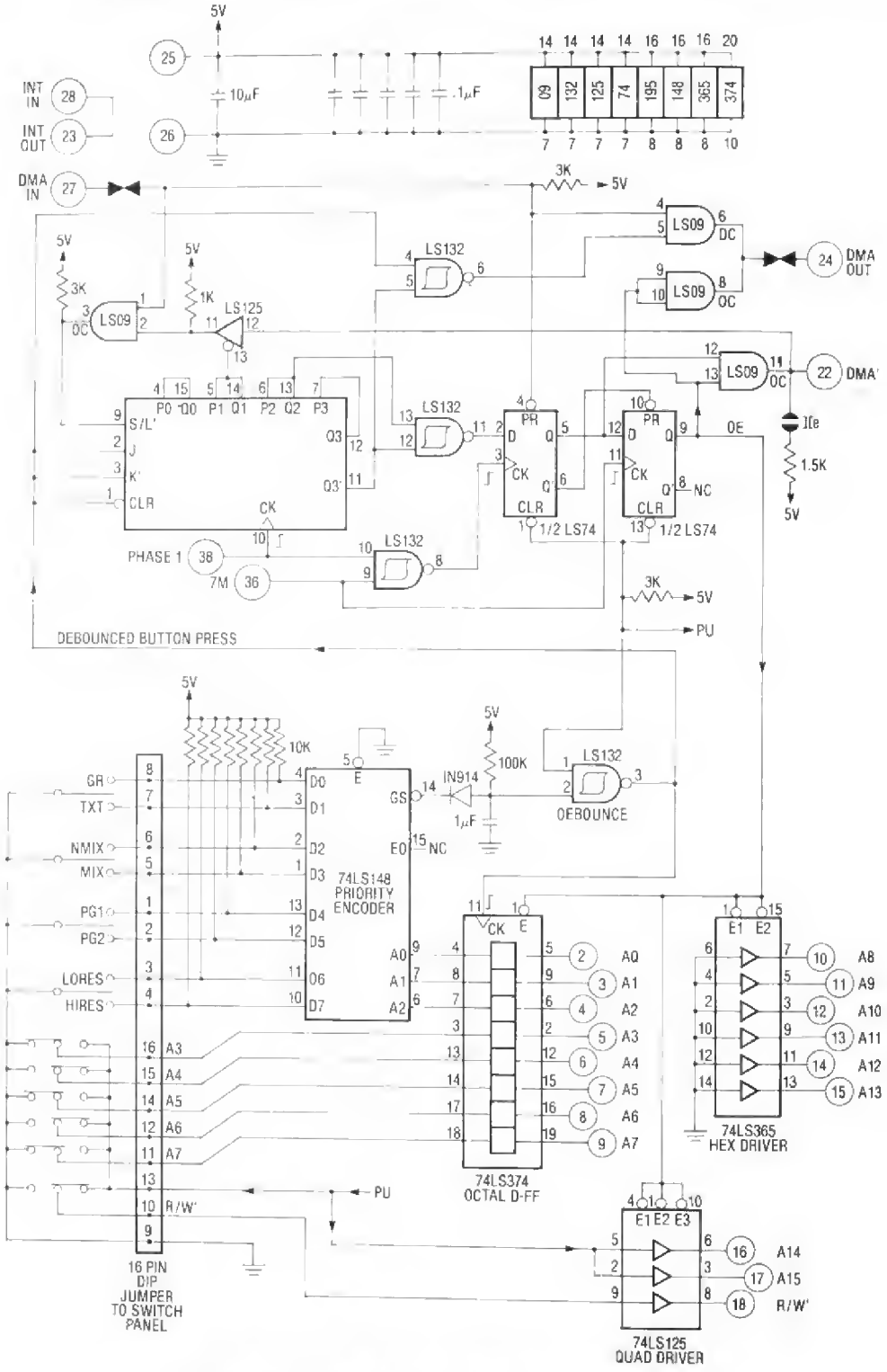


Figure 4.7 Schematic: D MAnual Controller.

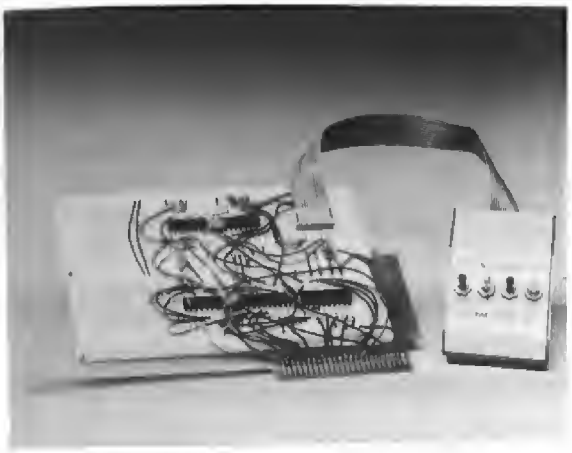


Figure 4.8 The Breadboarded Manual Controller.

signal falls and rises just after PHASE 1 rises. The cycle is further delayed for one half of a 7M period in the first half of the 74LS74. The resulting signal at pin 5 of the LS74 represents the DMA cycle.

When the DMA cycle signal at pin 5 of the 74LS74 goes low, the DMA' line is brought low. Half of a 7M period later, the data enable signal (LS74-9) of D MAnual Controller's address bus drivers drops low. This delay allows the MPU to be isolated from the address bus before any attempt is made to control the address bus. At the end of the DMA cycle, the opposite order is observed. Because of propagation delays in the LS09 feeding the DMA' line and motherboard IC C11, control of the address bus is released before the MPU address bus driver is enabled.

D MAnual Controller supports the DMA priority chain so it can operate with some other peripherals which perform DMA. It respects the DMA priority input and will delay its access until a higher priority device has finished its DMA. It also will steal cycles from a lower priority DMA card if that card respects its priority input.

Supporting the DMA priority chain is a little difficult, because it is the most abused protocol since "do unto others..." Apple abused it by not publishing a protocol, and by using the DMA priority chain in the 12K firmware card. Microsoft abused it in their older, DMA based Z80 *Softcard* by requiring higher priority devices to wait several cycles after bringing the priority line low before the Softcard will get off the bus. Yet when the *Softcard* takes over the bus itself, it gives lower priority DMA cards no similar consideration. Some DMA based MPU cards don't support the DMA priority chain at all. Other DMA cards support or ignore the DMA priority chain in ways which make sharing of the DMA capability unpredictable.

D MAnual Controller gets around the unpredictability of other DMA card designs by monitoring the DMA' line and delaying its own DMA cycle if DMA' is being held low by another card. This should work with other DMA designs which make any attempt to support the priority chain. Here are some ways to install D MAnual Controller with other DMA related cards:

1. **Z80 Softcard.** The Apple IIe Z80 *Softcard* resides in the auxiliary slot and does not perform DMA. D MAnual Controller does not interfere with this card in any way. The older

Table 4.5 Operation of Soft Switches from D MAnual Controller.

AAAA A RW 7654 3	FUNCTION	BUTTON 0/1	BUTTON 2/3	BUTTON 4/5	BUTTON 6/7
W 0000 0	IIe MEMORY MANAGE	80STORE	RAMRD	RAMWRT	INTCXROM
W 0000 1	IIe MEMORY MANAGE	ALTZP	SLOTC3ROM	80COL	ALTCHARSET
X 0010 X	CASSETE OUT TOGGLE	----------PUSH ANY BUTTON----------			
X 0011 X	SPEAKER TOGGLE	----------PUSH ANY BUTTON----------			
X 0100 X	C040 STROBE	----------PUSH ANY BUTTON----------			
X 0101 0	SCREEN MODE CTRL	GR/TXT	NMIX/MX	PG2/PG1	LORES/HIRES
X 0101 1	ANNUNCIATOR CTRL	AN0	AN1	AN2	AN3
X 0101 1	DOUBLE-RES GRAPHIX	--------	---------	----------	ENA/DSBL
X 1000 X	FIRMWARE CARD CTRL	ENA/DSBL	ENA/DSBL	ENA/DSBL	ENA/DSBL
X 1110 0	DISK HEAD CONTROL	PHASE-0	PHASE-1	PHASE-2	PHASE-3
X 1110 1	DISK CONTROL	OFF/ON	DRIVE 1/2	SHIFT/LOAD	READ/WRITE

Table 4.6 Selection of 16K RAM Card or IIe High RAM from D MAnual Controller.

AAAA A RW 7654 3	FUNCTION	BUTTON Ø OR 4	BUTTON 1 OR 5	BUTTON 2 ØR 6	BUTTON 3 OR 7
R 1000 Ø	BANK 2 CTRL	READON - WRTOFF WRTCOUNT=Ø	READOFF WRTCOUNT+1	READOFF - WRTOFF WRTCOUNT=Ø	READON WRTCOUNT+1
W 1000 Ø	BANK 2 CTRL	READON - WRTOFF WRTCOUNT=Ø	READOFF WRTCOUNT=Ø	READOFF - WRTOFF WRTCOUNT=Ø	READON WRTCOUNT=Ø
R 1000 1	BANK 1 CTRL	READON - WRTOFF WRTCOUNT=Ø	READOFF WRTCOUNT+1	READOFF - WRTOFF WRTCOUNT=Ø	READON WRTCOUNT+1
W 1000 1	BANK 1 CTRL	READON - WRTOFF WRTCOUNT=Ø	READOFF WRTCOUNT=Ø	READOFF - WRTOFF WRTCOUNT=Ø	READON WRTCOUNT=Ø

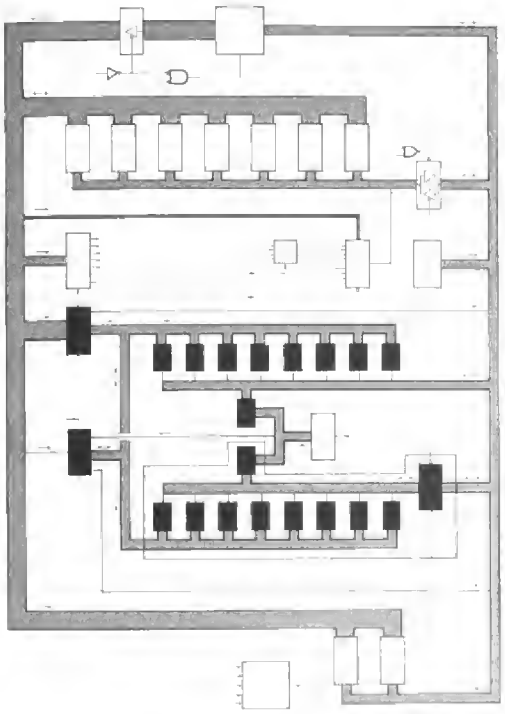
Softcard which plugs into a peripheral slot does perform DMA. If you use this type of *Softcard*, then install D MAnual Controller in a higher priority slot. It will steal a cycle from the *Softcard* without affecting its operation. Switch S2 on the *Softcard* must be **on** for this configuration to work. Solder the IIe jumper on D MAnual Controller if operating in an Apple IIe, but leave the IIe jumper unsoldered if operating in an Apple II.* Other peripheral slot MPU cards which support the DMA priority chain should work in this configuration.

2. **Firmware Card.** Apple's 12K firmware card uses the DMA priority line even though it does not perform DMA. Since D MAnual Controller only needs a single cycle, it will work with a firmware card enabled in a higher priority slot. It just waits until the MPU accesses a non-ROM address, then steals a cycle. If a secondary MPU card like the *Softcard* happens to be in a lower priority slot, the firmware card can interfere with that card's operation. This can be prevented by opening the DMA IN jumper on D MAnual Controller. Lower priority firmware cards are not interfered with by D MAnual Controller, because the Controller does not generate addresses in the firmware card range. The firmware card rules also apply to the SCRG *quickLoader* and other memory expansion cards which use the DMA priority chain to prioritize multiple card configurations.
3. **Disk or Cassette I/O.** Disk and cassette I/O in the Apple are normally performed in precise timing loops. Any DMA device which is activated in the midst of such loops will interfere with them and the associated data transfer. Therefore, you should never operate a pushbutton of D MAnual Controller while cassette or disk I/O is being performed, especially during write operations. Some hard disk or eight inch floppy disk Apple interfaces are DMA based. Any such devices should probably be mounted in a higher priority slot than D MAnual Controller, so that if they support the DMA priority chain, the integrity of disk data transfer will be insured.
4. **DMA cards which do not support the priority chain.** Cards like these are like citizens who do not meet their responsibilities to society. Do not operate the pushbuttons of D MAnual Controller when such cards are active. Only one card at a time can perform DMA in the Apple.

Readers who wish to are encouraged to build D MAnual Controller for their own use. They may also purchase the Controller, assembled and tested. The Controller is being manufactured by the Southern California Research Group. Readers of this book may order a D MAnual Controller by contacting:

D MAnual Controller
Southern California Research Group
13793 Christian Barrett Drive
Moorpark CA 93021
(805) 529-2082
(800) 821-0774 In CA, for orders only
(800) 635-8310 Outside CA, for orders only

*In the Apple IIe, 3300-ohm pull-up resistors are used on the wire-OR lines instead of 1000-ohm resistors. This results in a switching time which is too slow for stealing cycles from the *Softcard*. Soldering the IIe jumper will speed switching time by paralleling the 3300-ohm motherboard resistor with a 1500-ohm resistor. If other DMA card designers begin to add this 1500-ohm resistor, the IIe jumper should only be connected on one of the cards. Incidentally, D MAnual Controller is a good candidate for installation in Slot 3 of the Apple IIe, because it will work in Slot 3, even though a RAM/80-column card is plugged into the auxiliary slot.



chapter 5

RAM and Memory Management

One might think that RAM and its associated circuitry should be a relatively easy subject. You write to it and read from it. What else is there?

Well, the **MPU** does write to and read from RAM, but the **video scanner** reads from RAM too. Additionally, both the MPU and video scanner access **auxiliary card RAM**.* And then there is the 64K dynamic RAM chip with its ROW address, COLUMN address, and refresh requirement. Add all of this to the most involved **bank switching** scheme ever imagined and you have a lot of functional and operational complexity.

When RAM is accessed by the MPU, motherboard circuitry must activate signals which tell the RAM chips to pass data to or receive data from the data bus. Control of Apple IIe data bus communication is the task of the MMU. An MPU program configures the Apple memory by setting up MMU soft switches, and when the MPU accesses an Apple IIe device, an MMU data bus management signal either directly or indirectly activates the device. Apple refers to this broad control function as **memory management**, and as will be seen, managing RAM in the Apple IIe is the major part of the task of managing memory.

In this chapter, we will examine the requirements of the 64K dynamic RAM chip, and the ways in which they are met in the Apple IIe. We will also see how the MMU monitors the address bus to manage the overall configuration of the Apple IIe memory map.

*Most discussion of RAM in *Understanding the Apple IIe* assumes that a 64K RAM card is installed in the auxiliary slot.

THE 64K DYNAMIC RAM CHIP

64K RAM chips are 65,536 **bit** read/write memories. As is indicated in the bus structure diagram in Figure 2.7, it takes eight chips to make up the 65,536 bytes of read/write memory on the Apple IIe motherboard. This standard chip is available from a number of manufacturers in a variety of speeds. With a 2 MHz access rate, the Apple does not put a particularly stringent speed requirement on its RAM.

The RAM chip is a 16-pin device requiring two power supply inputs, +5V and ground. Figure 5.2

shows the pin assignments of the 64K RAM chips (see RAM chip F6). There is a data input to accept write data, a tri-state data output to transfer read data, and a R/W' control input to identify read and write cycles.

It takes 16 bits to address 64K of RAM, but there are only eight address inputs to the RAM chip. The 16-bit address must be multiplexed onto the 8-bit RAM address input lines in the form of a **ROW address** followed by a **COLUMN address**. Think of the 65,536 memory cells as lying in a 256 x 256 matrix.* The first 8-bit address input to a RAM chip specifies which ROW the addressed cell lies in, and the second 8-bit address specifies the COLUMN. RAS' falling clocks the ROW address to RAM. CAS' falling clocks the COLUMN address to RAM and initiates the read or write action.

Figure 5.1 shows the **timing generator** signals which control RAM access in the Apple. The nature of these signals is dictated by 64K dynamic RAM chip requirements, 1 MHz 6502 timing requirements, and the alternating access between the MPU and the video scanner. PHASE 0 and RAS' provide the timing reference for scanner ROW/COLUMN addressing and for MPU ROW/COLUMN addressing. RAS' and CAS' are applied directly to the RAS' and CAS' inputs of all of the motherboard RAM chips, but CAS' does not fall during PHASE 0 unless the MPU is accessing motherboard RAM. On the 64K auxiliary RAM card, RAS' is applied to the RAM chip RAS' input, and Q3 is applied to the RAM chip CAS' input.

RAS' falling clocks the ROW address to the RAM chip. The address input to the chip must contain the ROW address when RAS' falls and the COLUMN address when CAS' falls. Placing the correct addressing signals at the address input to the RAM chips is the function of the **RAM address multiplexing** circuitry in the IOU and the MMU. CAS' initiates the data transfer by dropping low after RAS' has already dropped low. Motherboard RAM must be capable of responding to a read access within 374 nanoseconds of RAS' falling and within 234 nanoseconds of CAS' falling. Auxiliary card RAM must be capable of responding with read data within 356 nanoseconds of RAS' falling and within 147 nanoseconds of Q3 falling. The motherboard and auxiliary card requirements are met by 200-nanosecond or faster 64K dynamic RAM chips, which means they are met by any 64K dynamic RAM chips that are generally available.

*As mentioned in Chapter 2, this book refers to a unit of memory which stores a bit of information as a **cell**. Each RAM chip has 65,536 cells, and is capable of storing 65,536 bits of information. The eight associated cells which store a byte of information in the Apple are referred to as a memory **location**. The Apple IIe has 65,536 RAM locations on the motherboard.

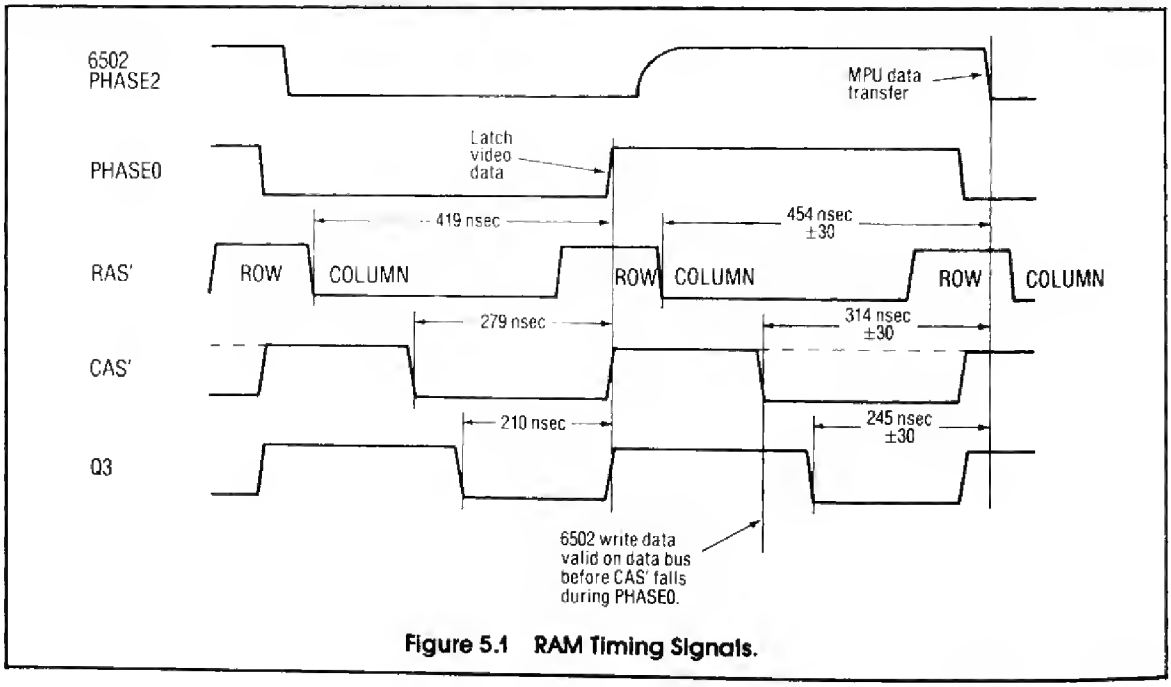


Figure 5.1 RAM Timing Signals.

Dynamic RAM must be periodically refreshed for it to operate properly. The refresh requirement of 64K chips is that each of the 256 possible ROW addresses must be accessed every two milliseconds, or 500 times a second.* This can be accomplished in RAS'/CAS' cycles or in RAS'-only cycles. It is accomplished in RAS'/CAS' cycles during PHASE 1 in the Apple IIe. The refresh requirement in the Apple is met in the process of scanning RAM for video output.

RAM CONNECTIONS IN THE APPLE IIe

The general flow of RAM data was discussed in Chapter 2. Figure 2.7, the bus structure diagram, should be reviewed to reinforce in your mind how RAM is tied into the overall scheme of things in the Apple. The basic features of RAM connections in the Apple are:

1. Motherboard RAM and auxiliary card RAM are each made up of eight 64K RAM chips. Each RAM chip is associated with one line of the data bus, and eight chips are capable of storing 65,536 bytes of data.
2. Motherboard RAM data input and output are tied directly to the data bus for MPU writing, MPU reading, and latching in the motherboard video latch.
3. Auxiliary card RAM data input and output are connected to the auxiliary card video latch and through a bidirectional driver to the data bus. The bidirectional driver isolates auxiliary card RAM from the data bus any time the MPU is not writing to or reading from auxiliary card RAM.
4. The latched video data is routed to the video generator for processing.
5. The RAM address input is multiplexed among the address bus ROW address (MMU), address bus COLUMN address (MMU), video scanner ROW address (IOU), and video scanner COLUMN address (IOU).

Figure 5.2 is a schematic diagram showing the connections of motherboard RAM, auxiliary card RAM, and the associated video data latches. The RAM chips are connected together in a way that reminds you of the wiring of the peripheral slots. The majority of the RAM lines are just strung from chip to chip. This includes the address input (RA0—RA7), +5V, ground, RAS', CAS' and RAM R/W' (motherboard), and Q3 and R/W'80 (auxiliary card).

*Some manufacturer's 64K RAM chips have a 128-cycle requirement that is easier to meet. The Apple IIe meets both the 128- and the 256-cycle refresh requirements.

The RAM connections support the demands of MPU read/write access to motherboard or auxiliary card RAM, alternating with simultaneous video scanner read access to both motherboard and auxiliary card RAM. During PHASE 1, the video scanner drives display data from motherboard RAM to the data bus and from auxiliary card RAM to the auxiliary card RAM data bus. This video data is saved in the motherboard and auxiliary card video data latches when PHASE 0 rises. During PHASE 0, the MPU reads or writes to motherboard RAM or auxiliary card RAM or another Apple device. Data transfer to or from the MPU occurs shortly after PHASE 0 falling when the 6502 PHASE 2 clock falls.

The RAM access scheme is complex, but the hardware implementation is compact. The critical control elements are the CASEN' and EN80' signals from the MMU, the read/write control and CAS' inputs to the RAM chips, and the enable/isolate input to the auxiliary card data bus driver.

The motherboard RAM R/W' signal is not the same as system R/W' from the 6502—it is system R/W' gated by PHASE 1 low (PHASE 0 high). The 6502 R/W' line drops low some time during PHASE 1 of write cycles. This would interfere with video scanner reading if 6502 R/W' were connected directly to RAM. Another way of looking at this is that the video scanner controls the motherboard RAM address and RAM R/W' during PHASE 1. The video scanner always reads, never writes.

CAS' from the timing HAL is connected to the motherboard RAM CAS' input (RAM pin 15), and RAM chip operation is such that when CAS' falls after RAS', data is passed in or out depending on RAM R/W'. As a result, motherboard RAM communicates with the MPU when the MMU brings CASEN' low, and consequently allows CAS' to fall during PHASE 0. The MMU brings CASEN' low when the MPU accesses an address that is configured in the MMU for motherboard RAM response. CAS' always falls during PHASE 1, so video data is always passed from motherboard RAM to the data bus during PHASE 1.

Auxiliary card RAM communication with the MPU is controlled differently than motherboard RAM communication. Q3 is tied to the auxiliary card RAM chip CAS' input, and Q3 always falls during PHASE 0 and PHASE 1. Therefore, auxiliary card RAM chips pass data in or out twice every MPU cycle. The RAM chip data, however, is isolated from the motherboard data bus by the 74LS245 bidirectional driver unless EN80' from the MMU is low. Operation of the LS245 is such that when its

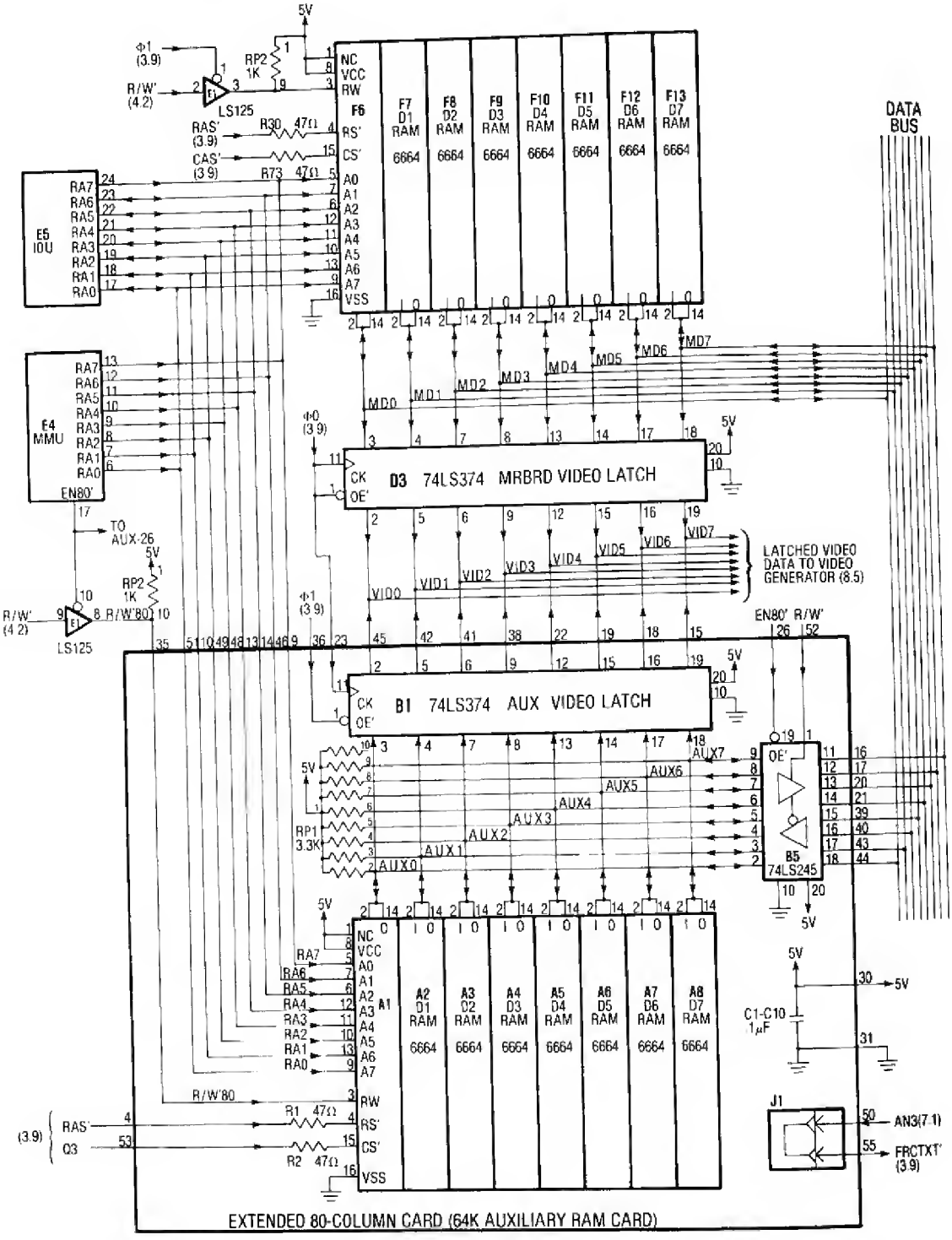


Figure 5.2 Schematic: Apple IIe RAM Connections.

enable input (EN80') is high, the driver presents a high impedance to both the motherboard data bus and the auxiliary card RAM data bus. EN80' is never low during PHASE 1, and it goes low during PHASE 0 only when the MPU is accessing an address that is configured in the MMU for auxiliary card RAM response.

The direction control input of the LS245 driver is R/W', so direction of the driver is correct when the MPU communicates with auxiliary card RAM. The read/write control of the RAM chips is R/W'80, which is the same as R/W' except that it can only go low when EN80' is low. Therefore, the MPU writes to auxiliary card RAM when EN80' is low and R/W' is low. Since EN80' is PHASE 0 gated, the video scanner access to auxiliary card RAM is always a read access, just as it is with motherboard RAM.

There are some subtle but important points about the handling of RAM output in the Apple. The data bus is available at the end of PHASE 1 because the MPU never controls the data bus at this time, not even during write cycles. The MPU controls the data bus for the second half of PHASE 0 and slightly beyond during write cycles. But nothing in a write cycle prevents the video scanner from reading RAM, and nothing prevents the RAM output from being latched for processing in the video generator. Therefore, write cycles do not cause random flicker in the Apple's video display.

Another interesting point is that data from the video scanner access to motherboard RAM is always available on the data bus when PHASE 0 rises. This video data is also passed through the peripheral slot bidirectional driver to the peripheral slots when R/W' is high or for a very short period after the beginning of PHASE 0 when R/W' is low. This means that motherboard video data can be read by peripheral cards, and many conceivable peripheral designs could make use of this data. A programmable video scanner simulator is one such design.

In SINGLE-RES or DOUBLE-RES display mode, latched auxiliary card RAM data is available on the video data bus during PHASE 0, and latched motherboard RAM data is available on the video data bus during PHASE 1. It is in the timing generator, not RAM, that SINGLE-RES and DOUBLE-RES processing are different from each other. In SINGLE-RES processing, no LDPS' pulse is generated during PHASE 0 to load the auxiliary video data into the video generator, so only the motherboard display map is processed. In DOUBLE-RES processing, LDPS' is generated during both PHASE 0 and PHASE 1, and both the motherboard and auxiliary card display maps are processed. Also, VID7M variations are such that video dot patterns are shifted out twice as fast in DOUBLE-RES and LORES40 processing as they are in TEXT40 and HIRES40 processing.

A point about Figure 5.2 which the reader may find confusing is the labeling of the multiplexed RAM address bus and the address inputs to the RAM chips. RA0 of the bus goes to A7 of the RAM chip; RA1 of the bus goes to A6 of the RAM chip; etc. Shouldn't RA0 go to A0 of the RAM chip? Well, that would have been logical, but it's not very important. Generally, the address-line labels on a RAM chip are arbitrary and have no operational significance.* The RAM works as long as each address bit is routed to one of the address inputs. In this case, the RAM chip labeling provided a natural reference for labeling the RAM address bus and the RAM address output pins of the IOU and MMU. For reasons unknown, Apple did not use this natural reference.

RAM ADDRESS MULTIPLEXING

Portions of both the IOU and MMU are devoted to RAM address multiplexing. Together, these multiplexing circuits make up a 4 to 1 RAM address multiplexor. The MMU develops the MPU ROW and COLUMN address from the address bus, and the IOU develops the video ROW and COLUMN address from the video scanner state and the Apple display mode.

RAM address multiplexing functions are summarized in Figure 5.3. In both the MMU and the IOU, RAS' high selects the ROW address input, and RAS' low selects the COLUMN address input. The MMU multiplexed address is gated to the RAM address bus during the last 14M period of PHASE 1 and the first four 14M periods of PHASE 0. The IOU multiplexed address is gated to the RAM address bus during the last 14M period of PHASE 0 and the first four 14M periods of PHASE 1. This gating ensures that, after propagation delay, the MPU address is valid at the RAM chips when RAS', CAS', and Q3 fall during PHASE 0, and the video scanner address is valid at the RAM chips when RAS', CAS', and Q3 fall during PHASE 1.

MMU multiplexing functions are a straightforward translation of the 16-bit MPU address into 8-bit ROW and COLUMN addresses. The only complication is that the A12 input to RA4-COL is forced

*An exception is systems in which only A0—A6 of the 64K chip are refreshed. A0—A7 of the RAM chips in the Apple IIe are refreshed, so this has no significance in the Apple IIe.

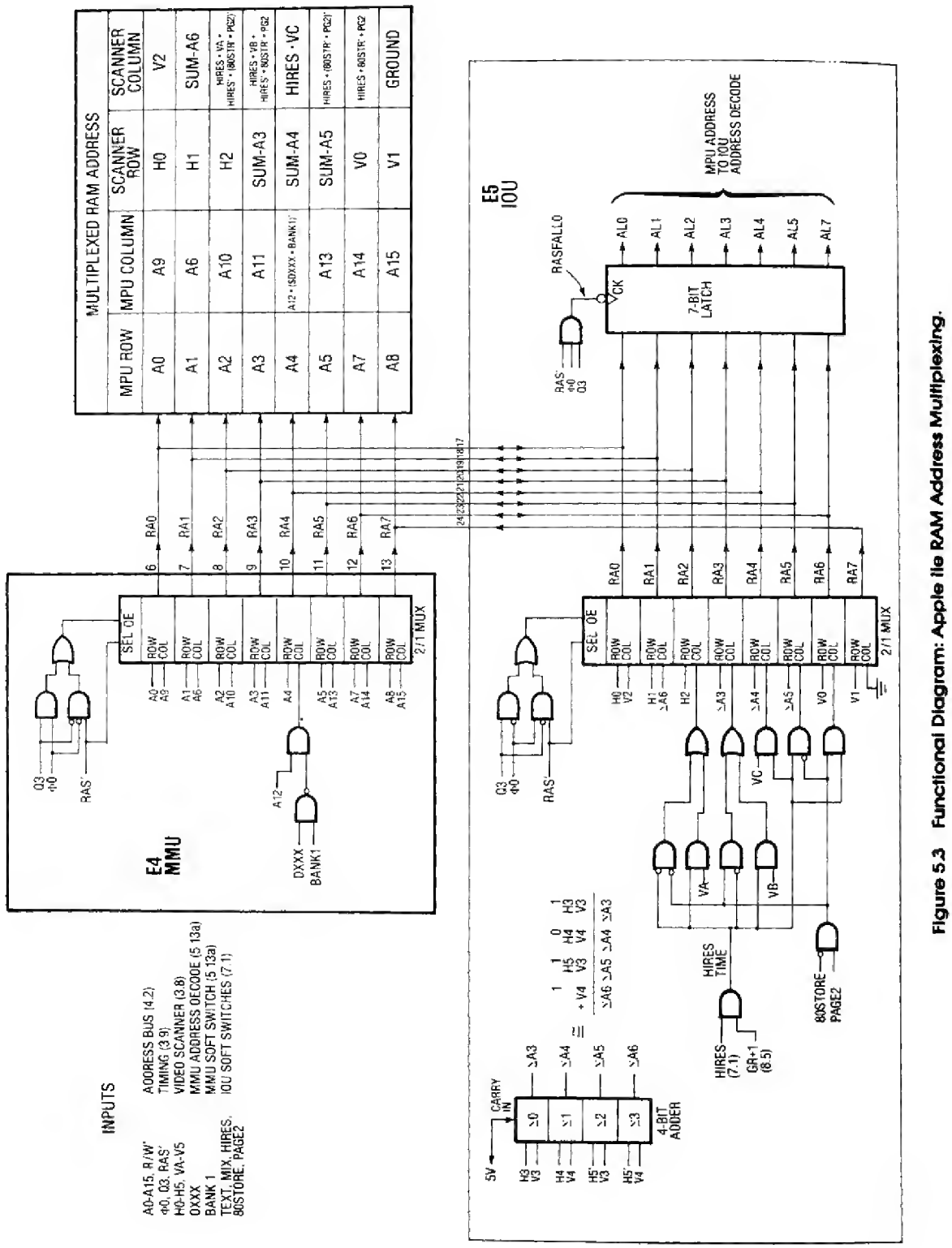


Figure 5.3 Functional Diagram: Apple IIe RAM Address Multiplexing.

low if an address in the $DXXX range is on the address bus and the BANK1 soft switch is set. Forcing A12 low changes $DXXX to $CXXX, and allows the MPU to access the bank 1 area of high RAM.

IOU multiplexing functions are more complex than those of the MMU. This is because the video RAM address is not simply a multiplexed form of the video scanner outputs. Instead, it is a combination of the Apple display mode and the video scanner output with special offset logic to facilitate the scanning of the Apple's unnatural 40-byte display line width. The display mode, which is set up via IOU soft switches, is an extension of the video scanner when it comes to addressing RAM. Generally, the display mode controls the high order address bits, thus determining what areas of RAM the video scanner accesses.

The HIRES TIME signal which is used in RAM addressing is a product of the video scanner state and the IOU display mode control soft switches. HIRES TIME is high when the Apple is in HIRES-GRAPHICS-NO MIX mode, or in HIRES-GRAPHICS-MIX mode when V4 • V2 is false. The V4 • V2 gating switches the scanned memory over to TEXT memory for four lines of text at the bottom of the screen. Naturally, the MIXED mode requires switching between GRAPHICS and TEXT in sync with the video scanner.

The PAGE2 and 80STORE inputs to Figure 5.3 are IOU soft switches. When set, PAGE2 selects secondary display memory pages for scanning. 80STORE, when set, overrides the effect of PAGE2 on memory scanning, thus inhibiting display of screen page 2. The 80STORE, PAGE2, and HIRES soft switches are also implemented in the MMU where they are used for switching between access to motherboard and auxiliary card RAM.

It should be noted that the *Apple II Reference Manual for IIe Only* says that 80VID, not 80STORE, is the soft switch which affects RAM addressing. This is an area in which the reference manual is inconsistent, using the terms 80STORE, 80VID, and 80COL to refer to what are, in fact, only two soft switches (W$C000/W$C001 and W$C00C/W$C00D). To avoid confusion, I suggest you change 80VID to 80COL throughout Apple's reference manual. On pages 3 and 4 of Apple's schematic, change 80VID' at pin 6 of the IOU and pin 25 of the auxiliary slot to 80COL'.

The soft switch that overrides PAGE2 in the IOU is 80STORE (W$C000/W$C001), not 80COL (W$C00C/W$C00D). The operational difference is great. If 80COL was the RAM addressing input, then there would be only one displayable page of DOUBLE-RES graphics. This, though, is not the case. By resetting 80STORE and setting PAGE2, a programmer can select the $800—$BFF area for TEXT/LORES display or the $4000—$5FFF area for HIRES display, even in the DOUBLE-RES modes. Furthermore, if 80STORE is set, the secondary pages cannot be displayed.

The RAM address inputs are selected from the address bus, the video scanner state, and the display mode. The MULTIPLEXED RAM ADDRESS table in Figure 5.3 shows the way address bus lines and video scanner output lines are assigned to RA0—RA7 ROW and RA0—RA7 COLUMN. There are some significant aspects to these address assignments:

1. The scanner low order bits are assigned to RAM ROW address inputs so the RAM will be refreshed by the video scanner.
2. The address bus bit which controls a given RAM address will be equivalent to the scanner bit which controls the same RAM address. For example, A0 controls RA0 during an MPU ROW access, and H0 controls RA0 during a scanner ROW access. This means that A0 and H0 perform the equivalent RAM addressing function.
3. The address bus low order bits are assigned to RAM ROW address inputs for correct equivalency to scanner addressing bits. It happens that this assignment results in A0—A5 and A7 being available to the IOU as part of the MMU ROW address. The only other address input the IOU needs is A6, and A6 is input directly to the IOU from the address bus.
4. The addressing of RAM is the same in DOUBLE-RES as it is in SINGLE-RES display modes.

Table 5.1 shows the equivalent address bus/video scanner address bits. You can use this table to take any screen mode and video scanner state and convert them to an equivalent MPU address. If you store a byte of data at the equivalent MPU address, it will be driven out of RAM during PHASE 1 when the video scanner reaches the chosen state.

The Arithmetic of Video Scanner Memory Addressing

If the Apple isn't famous for the encrypted nature of its screen memory addressing, it should be. The programmer has a very heavy burden in computing or looking up seemingly illogical addresses. This goes all the way back to the original design of the Apple II, since Apple IIe display memory addresses are the same as those of the older computer.

Table 5.1 MPU/Scanner Equivalent Address Bits.

MPU	VIDEO SCANNER	
A0	H0	
A1	H1	
A2	H2	
A3	SUM-A3	
A4	SUM-A4	
A5	SUM-A5	
A6	SUM-A6	
A7	V0	
A8	V1	
A9	V2	
A10	HIRES • VA	TEXT/LORES • (PAGE2 • 80STORE')'
A11	HIRES • VB	TEXT/LORES • PAGE2 • 80STORE'
A12	HIRES • VC	—
A13	HIRES • (PAGE2 • 80STORE')'	—
A14	HIRES • PAGE2 • 80STORE'	—
A15	—	

There is logic to the Apple screen memory addressing. It is the logic of binary manipulation. The way to understand it is to look at the Apple from the designer's viewpoint. In 1975, how would you have gotten the Apple to display HIRES color graphics, LORES color graphics, and 40 columns of text?

Forty columns? Two strikes against you to start with. Didn't Wozniak ever hear of powers of two? Digital computers are based in binary numbers. Use 32, 64, or 128 columns. This is as bad as the guys who designed 80-column typewritten page widths and 10-digit humans.

The problem is that you want to address memory sequentially with the output lines of the video scanner. If the Apple line width had been 32 columns, you could just tie H0—H5 and V0—V4 directly to a 4 to 1 address multiplexor. Memory would be very neatly divided up into 32 x 24 bytes.

Upgrading the display from 32 to 40 columns makes the scanner address assignments less straightforward. You can achieve a 40-column display by tying H0—H5 directly to the 4 to 1 address multiplexor, and this would create an easy hardware connection. But there would be unused gaps in memory 24 bytes long for every 40 bytes used. This would waste 576 bytes of memory in TEXT/LORES modes and 4608 bytes in HIRES mode. What good are 4608 bytes of memory divided up into 192 noncontiguous groups of 24 bytes?

In the Apple, it was accepted that there would be some waste of memory caused by the 40-character lines, but the waste was minimized at the expense of a little hardware complexity. Instead of using 40 bytes out of each 64-byte memory segment, 120 bytes out of each 128-byte memory segment are used. This creates eight bytes of wasted memory for every three horizontal scans in HIRES or every three lines of characters in TEXT. As a result, there is a total wastage of 512 bytes in HIRES and 64 bytes in TEXT/LORES.

How do you implement this in hardware? Screen memory is divided into **128-byte segments** (see Figure 5.4). Each segment is divided into the **FIRST 40**, the **SECOND 40**, the **THIRD 40**, and eight bytes of no man's memory (**UNUSED 8**). It so happens that the displayed television scan is neatly divided into three sections by V3 and V4 from the video scanner as follows:

V4' V3' —Top third of television screen
V4' V3 —Middle third of television screen
V4 V3' —Bottom third of television screen
V4 V3 —Undisplayed (VBL)

Because the three displayed portions of the screen can be so easily detected, they are mapped into the three 40-byte sections of each 128-byte memory segment as follows:

LOCATION ON TV SCREEN	LEAST SIGNIFICANT BITS OF ADDRESS
Top	0000000—0100111 (FIRST 40)
Middle	0101000—1001111 (SECOND 40)
Bottom	1010000—1110111 (THIRD 40)

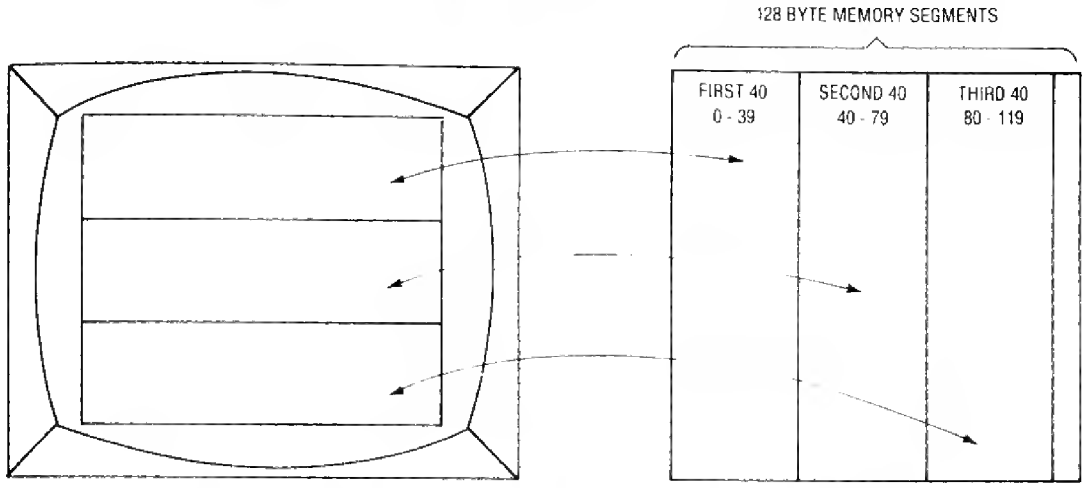


Figure 5.4 128-Byte Video Memory Segments Consist of Three 40-Byte Sections, Each Mapped Into a Different Part of the Video Screen.

It can be seen in the binary representations of the 40-byte address sections, that the lower three bits cross over from 111 to 000 at all section boundaries. This means that these three bits can be identically addressed in the FIRST 40, SECOND 40, or THIRD 40. For example, the lowest three bits of the address of the left most character in any section is 000. For this reason, H0, H1, and H2 are direct address inputs to the IOU address multiplexor and are address equivalents of A0, A1, and A2.

The next four address bits are different depending on the 40-byte section that is being addressed. They are 0000 through 0100 in the FIRST 40, 0101 through 1001 in the SECOND 40, and 1010 through 1110 in the THIRD 40. These four bits are addressed by H5-H4-H3 plus an offset. The offset value is selected by V4 and V3 to place the scanned memory address in the FIRST 40, SECOND 40, or THIRD 40 of the current 128-byte segment. The offset is added to H5-H4-H3 in a 4-bit adder in the IOU, and the four bits of the resulting SUM become the scanning address bits equivalent to A3, A4, A5, and A6. This book refers to the SUM bits as SUM-A3, SUM-A4, SUM-A5, and SUM-A6.

There are eight states of H5-H4-H3 but only five of the states are displayed. 000 through 010 are undisplayed and occur during the right margin, horizontal retrace, and left margin of the television scan. 011 is the first displayed count, and when H5-H4-H3 reaches 011, it is time to address the first byte of a 40-byte section. Suppose you built the following summing circuit:

```
                  H5        H4        H3
+   V4            V3        V4        V3
SUM-A6      SUM-A5    SUM-A4    SUM-A3
```

This would create the three offsets 0, 101, and 1010, which are 40-byte offsets. This circuit would work, but it would make screen memory address assignments even more complex than they are for the Apple programmer. Since the display starts at H5-H4-H3 = 011, we need to subtract 011 from the offsets 000, 101, and 1010 to make the FIRST 40 start at a natural 128-byte segment boundary. The required offsets are 1101, 010, and 111 (−3, 2, and 7 in decimal). You need to address A6-A5-A4-A3 with the values H5-H4-H3 minus 011 in the FIRST 40, H5-H4-H3 plus 010 in the SECOND 40, and H5-H4-H3 plus 111 in the THIRD 40. These sums are cleverly created in the Apple by the following addition:

```
                                    1
    H5'       H5'       H4        H3
+   V4        V3        V4        V3
SUM-A6    SUM-A5    SUM-A4    SUM-A3
```

H5'-H5'-H4-H3 is equal to H5-H4-H3 minus 100 in 4-bit signed binary arithmetic. 001 minus 100 is −011, so the needed offset is developed. It is easy to add 1 as a carry input to the 4-bit adder. The equivalent adding circuit is:

```
    1         1         Ø         1
              H5        H4        H3
+   V4        V3        V4        V3
SUM-A6    SUM-A5    SUM-A4    SUM-A3
```

TEXT/LORES Scanning

Beyond A6, scanning address assignments determine the memory blocks scanned in the various screen modes. V0, V1, and V2 are equivalent to A7, A8, and A9 in all screen modes. A TEXT/LORES screen memory page is made up of eight adjacent 128-byte segments. These eight adjacent segments are defined by V0, V1, and V2 in the scanner and by A7, A8, and A9 on the address bus. VA, VB, and VC play no part in addressing TEXT/LORES memory. Rather, the same 40-byte section of memory is scanned for eight adjacent horizontal television lines. It takes eight horizontal television lines to paint one line of text or two rows of LORES blocks. In video generation circuits internal and external to the IOU, VA, VB, and VC define which vertical part of a text character it is time to draw, and VC defines which of two LORES blocks it is time to draw.

In TEXT/LORES, A15, A14, A13, and A12 equivalents are false (low, ground, zip). The A10 equivalent is (80STORE' • PAGE2)', and the A11 equivalent is 80STORE' • PAGE2. This results in the memory scanned areas for TEXT/LORES shown at the bottom of this page.

Figure 5.5 is the TEXT/LORES displayed memory map. This map shows the same information as the maps of the *Apple II Reference Manual for IIe Only*, but there is a difference in layout. The reference manual maps accent the 24 lines of text or 48 lines of LORES blocks, but Figure 5.5 accents the division of screen memory into 128-byte memory segments. This should give the reader a second perspective from which to view the screen mapping.

In addition to the displayed memory locations, there is reason to know what areas of memory are being scanned while nothing is being displayed. This blanking time is defined by horizontal and vertical blanking gates generated in the IOU from video scanner outputs. HBL (Horizontal BLanking gate) is high during the right margin, horizontal retrace, and left margin of the Apple video display. VBL (Vertical BLanking gate) is high during the bottom margin, vertical retrace, and top margin of the Apple video display.

Knowledge of memory scanned during HBL and VBL has applications when software or hardware syncs to the video scan by detecting the scanned memory output on the data bus or peripheral data bus. This is possible in software by reading an address from which there is no data response. For example, if you zero out all scanned memory except for the bytes in the blanking period preceding a given horizontal display line, you can detect the beginning of that horizontal scan with the following loop.

```
PAGE1   EQU   $C054
WAIT    LDA   PAGE1
        BPL   WAIT
```

Some techniques of exploiting this capability are discussed in an application note at the end of this chapter. The point is that it is sometimes useful to know what areas of memory are being scanned during blanking periods.

Figure 5.6 is a TEXT/LORES map showing the areas of memory scanned during displayed and undisplayed periods. The layout is similar to the maps in the *Apple II Reference Manual for IIe Only*. The area of memory scanned previous to every horizontal display period is shown directly to the left of the memory scanned during that display period. The vertical blanking period is shown at the bottom. The considerations which determine the memory scanned during the blanking periods are as follows:

1. HBL scanned memory begins $18 bytes before display scanned memory. The HBL base address can be computed from the displayed base address using this Applesoft program sequence:

```
10 HBL = BASE-24
20 IF INT(HBL/128)<>INT(BASE/128)
   THEN HBL = HBL+128
```

   Step 20 of the above program is necessary because horizontal memory addressing wraps around at the 128-byte segment boundaries.
2. The first address of HBL is always addressed twice consecutively, because H0—H5 is in the all zero state for two consecutive scans.
3. During VBL (vertical blanking), V3 and V4 are both true. The horizontal offset sum becomes H5-H4-H3 minus 0100. This is almost the same as the top of the displayed screen (H5-H4-H3 minus 0011). The VBL base addresses are equal to the FIRST 40 base addresses minus eight

SCREEN MODE	BINARY	HEXADECIMAL
PAGE 1	0000 01XX XXXX XXXX	$0400-$07FF
PAGE 2, 80STORE	0000 01XX XXXX XXXX	$0400-$07FF
PAGE 2, 80STORE'	0000 10XX XXXX XXXX	$0800-$0BFF

	BASE ADDRESS	TOP SCREEN/ FIRST 40 LIN#	RANGE	MIDDLE SCREEN/ SECOND 40 LIN#	RANGE	BOTTOM SCREEN/ THIRD 40 LIN#	RANGE	UNUSED 8 RANGE
	$400	00	$400-$427	08	$428-$44F	16	$450-$477	$478-$47F
	1024		1024-1063		1064-1103		1104-1143	1144-1151
	$480	01	$480-$4A7	09	$4A8-$4CF	17	$4D0-$4F7	$4F8-$4FF
	1152		1152-1191		1192-1231		1232-1271	1272-1279
	$500	02	$500-$527	10	$528-$54F	18	$550-$577	$578-$57F
	1280		1280-1319		1320-1359		1360-1399	1400-1407
	$580	03	$580-$5A7	11	$5A8-$5CF	19	$5D0-$5F7	$5F8-$5FF
PAGE 1	1408		1408-1447		1448-1487		1488-1527	1528-1535
	$600	04	$600-$627	12	$628-$64F	20	$650-$677	$678-$67F
	1536		1536-1575		1576-1615		1616-1655	1656-1663
	$680	05	$680-$6A7	13	$6A8-$6CF	21	$6D0-$6F7	$6F8-$6FF
	1664		1664-1703		1704-1743		1744-1783	1784-1791
	$700	06	$700-$727	14	$728-$74F	22	$750-$777	$778-$77F
	1792		1792-1831		1832-1871		1872-1911	1912-1919
	$780	07	$780-$7A7	15	$7A8-$7CF	23	$7D0-$7F7	$7F8-$7FF
	1920		1920-1959		1960-1999		2000-2039	2040-2047
	$800	00	$800-$827	08	$828-$84F	16	$850-$877	$878-$87F
	2048		2048-2087		2088-2127		2128-2167	2168-2175
	$880	01	$880-$8A7	09	$8A8-$8CF	17	$8D0-$8F7	$8F8-$8FF
	2176		2176-2215		2216-2255		2256-2295	2296-2303
	$900	02	$900-$927	10	$928-$94F	18	$950-$977	$978-$97F
	2304		2304-2343		2344-2383		2384-2423	2424-2431
	$980	03	$980-$9A7	11	$9A8-$9CF	19	$9D0-$9F7	$9F8-$9FF
PAGE 2	2432		2432-2471		2472-2511		2512-2551	2552-2559
	$A00	04	$A00-$A27	12	$A28-$A4F	20	$A50-$A77	$A78-$A7F
	2560		2560-2599		2600-2639		2640-2679	2680-2687
	$A80	05	$A80-$AA7	13	$AA8-$ACF	21	$AD0-$AF7	$AF8-$AFF
	2688		2688-2727		2728-2767		2768-2807	2808-2815
	$B00	06	$B00-$B27	14	$B28-$B4F	22	$B50-$B77	$B78-$B7F
	2816		2816-2855		2856-2895		2896-2935	2936-2943
	$B80	07	$B80-$BA7	15	$BA8-$BCF	23	$BD0-$BF7	$BF8-$BFF
	2944		2944-2983		2984-3023		3024-3063	3064-3071

Figure 5.5 TEXT/LORES Displayed Memory Map.

bytes using 128-byte wraparound subtraction. Example: $400 minus $8 gives $478, not $3F8.

4. Horizontal scanning wraps around at the 128-byte segment boundaries. Example: the address scanned before address $400 is $47F.

HIRES Scanning

Table 5.1 shows that HIRES video scanner addressing is identical to TEXT/LORES addressing on bits A0—A9 and A15. The differences in bits A10—A14 reflect the facts that HIRES memory is eight times as big as TEXT/LORES memory, and in a different location than TEXT/LORES memory.

During HIRES scanning, A13 is equivalent to (PAGE2 • 80STORE')', and A14 is equivalent to PAGE2 • 80STORE'. This results in a page 1 base address of $2000 and a page 2 base address of $4000. The effect of 80STORE, as in TEXT/LORES scanning, is to override the PAGE2 soft switch.

VA, VB, and VC are equivalent to A10, A11, and A12 in HIRES. This is the most important point, because it represents the great difference between HIRES and TEXT/LORES. In TEXT/LORES, 40 bytes contain the display intelligence for eight horizontal scans. In HIRES, 40 bytes contain the display intelligence for one horizontal scan. The HIRES scan must address a different 40-byte section every scan. This is accomplished by letting VA, VB, and VC affect the memory address in HIRES scanning.

Notice in Table 5.1 the oddity that VA, VB, and VC address higher order bits of memory than do V0, V1, and V2. This causes the base addresses of adjacent HIRES lines to be separated by 1024 bytes, rather than the logical 128 bytes. This extra complication of HIRES address computation could have been eliminated in the original Apple II design by the addition of one chip. It wasn't, so the user suffers an extra operational distraction in the Apple II and the IIe. One way to look at the HIRES memory layout is as eight adjacent areas with each area the memory equivalent of a single TEXT/LORES page (see Figure 5.7). VA, VB, and VC determine which of the eight areas is being addressed. As eight adjacent horizontal lines are scanned, one 64-byte (40 bytes displayed) section from each of the eight memory areas is scanned. As in TEXT/LORES, the top, middle, and bottom thirds of the screen are accompanied by memory scanning of the FIRST 40, SECOND 40, and THIRD 40 sections respectively.

One way to gain insight into the overall layout of HIRES memory is to run the following BASIC program:

```
10 HGR : POKE -16302,0 :
   POKE -16372,0 : REM HIRES40,
   NOMIX
20 FOR A = 8192 TO 16383
30 POKE A,255
40 FOR B = 0 TO 100 : NEXT B :
   REM DO IT SLOWLY
50 NEXT A : GO TO 10
```

This program fills the consecutive memory locations of HIRES, PAGE 1 with $FF slowly so that you can watch the screen fill.

Figure 5.8 is a HIRES displayed memory map accenting the division of screen memory into 128-byte memory segments. This figure was printed out using an Applesoft program listed in Appendix D. Like the TEXT/LORES map of Figure 5.5, this

		HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
	LIN NUM	PAGE 1	PAGE 2	11111111 01234567890ABCDEF01234567	PAGE 1	PAGE 2	1111111111111111222222222 01234567890ABCDEF01234567890ABCDEF01234567
	0	$400 1024	$800 2048	++++++++####++++++++	$400 1024	$800 2048	++++++++++++++++++++++++++++++++
	1	$480 1152	$880 2176	++++++++####++++++++	$480 1152	$880 2176	++++++++++++++++++++++++++++++++
	2	$500 1280	$900 2304	++++++++####++++++++	$500 1280	$900 2304	++++++++++++++++++++++++++++++++
SCREEN	3	$580 1408	$980 2432	++++++++####++++++++	$580 1408	$980 2432	++++++++++++++++++++++++++++++++
TOP	4	$600 1536	$A00 2560	++++++++####++++++++	$600 1536	$A00 2560	++++++++++++++++++++++++++++++++
	5	$680 1664	$A80 2688	++++++++####++++++++	$680 1664	$A80 2688	++++++++++++++++++++++++++++++++
	6	$700 1792	$B00 2816	++++++++####++++++++	$700 1792	$B00 2816	++++++++++++++++++++++++++++++++
	7	$780 1920	$B80 2944	++++++++####++++++++	$780 1920	$B80 2944	++++++++++++++++++++++++++++++++
	8	$428 1064	$828 2088	++++++++####++++++++	$428 1064	$828 2088	++++++++++++++++++++++++++++++++
	9	$4A8 1192	$8A8 2216	++++++++####++++++++	$4A8 1192	$8A8 2216	++++++++++++++++++++++++++++++++
	10	$528 1320	$928 2344	++++++++####++++++++	$528 1320	$928 2344	++++++++++++++++++++++++++++++++
SCREEN	11	$5A8 1448	$9A8 2472	++++++++####++++++++	$5A8 1448	$9A8 2472	++++++++++++++++++++++++++++++++
MIDDLE	12	$628 1576	$A28 2600	++++++++####++++++++	$628 1576	$A28 2600	++++++++++++++++++++++++++++++++
	13	$6A8 1704	$AA8 2728	++++++++####++++++++	$6A8 1704	$AA8 2728	++++++++++++++++++++++++++++++++
	14	$728 1832	$B28 2856	++++++++####++++++++	$728 1832	$B28 2856	++++++++++++++++++++++++++++++++
	15	$7A8 1960	$BA8 2984	++++++++####++++++++	$7A8 1960	$BA8 2984	++++++++++++++++++++++++++++++++
	16	$450 1104	$850 2128	++++++++####++++++++	$450 1104	$850 2128	++++++++++++++++++++++++++++++++
	17	$4D0 1232	$8D0 2256	++++++++####++++++++	$4D0 1232	$8D0 2256	++++++++++++++++++++++++++++++++
	18	$550 1360	$950 2384	++++++++####++++++++	$550 1360	$950 2384	++++++++++++++++++++++++++++++++
SCREEN	19	$5D0 1488	$9D0 2512	++++++++####++++++++	$5D0 1488	$9D0 2512	++++++++++++++++++++++++++++++++
BOTTOM	20	$650 1616	$A50 2640	++++++++####++++++++	$650 1616	$A50 2640	++++++++++++++++++++++++++++++++
	21	$6D0 1744	$AD0 2768	++++++++####++++++++	$6D0 1744	$AD0 2768	++++++++++++++++++++++++++++++++
	22	$750 1872	$B50 2896	++++++++####++++++++	$750 1872	$B50 2896	++++++++++++++++++++++++++++++++
	23	$7D0 2000	$BD0 3024	++++++++####++++++++	$7D0 2000	$BD0 3024	++++++++++++++++++++++++++++++++
	24	$478 1144	$878 2168	++++++++####++++++++	$478 1144	$878 2168	++++++++++++++++++++++++++++++++
	25	$4F8 1272	$8F8 2296	++++++++####++++++++	$4F8 1272	$8F8 2296	++++++++++++++++++++++++++++++++
	26	$578 1400	$978 2424	++++++++####++++++++	$578 1400	$978 2424	++++++++++++++++++++++++++++++++
VERTICAL	27	$5F8 1528	$9F8 2552	++++++++####++++++++	$5F8 1528	$9F8 2552	++++++++++++++++++++++++++++++++
BLANKING	28	$678 1656	$A78 2680	++++++++################	$678 1656	$A78 2680	################################
	29	$6F8 1784	$AF8 2808	++++++++####++++++++	$6F8 1784	$AF8 2808	++++++++++++++++++++++++++++++++
	30	$778 1912	$B78 2936	++++++++####++++++++	$778 1912	$B78 2936	++++++++++++++++++++++++++++++++
	31	$7F8 2040	$BF8 3064	++++++++####++++++++	$7F8 2040	$BF8 3064	++++++++++++++++++++++++++++++++

The last row of memory is scanned 14 consecutive times. All other rows are scanned 8 consecutive times.

HORIZONTAL SYNC

VERTICAL SYNC

Figure 5.6 TEXT/LORES Scanning Map Including Undisplayed Areas.

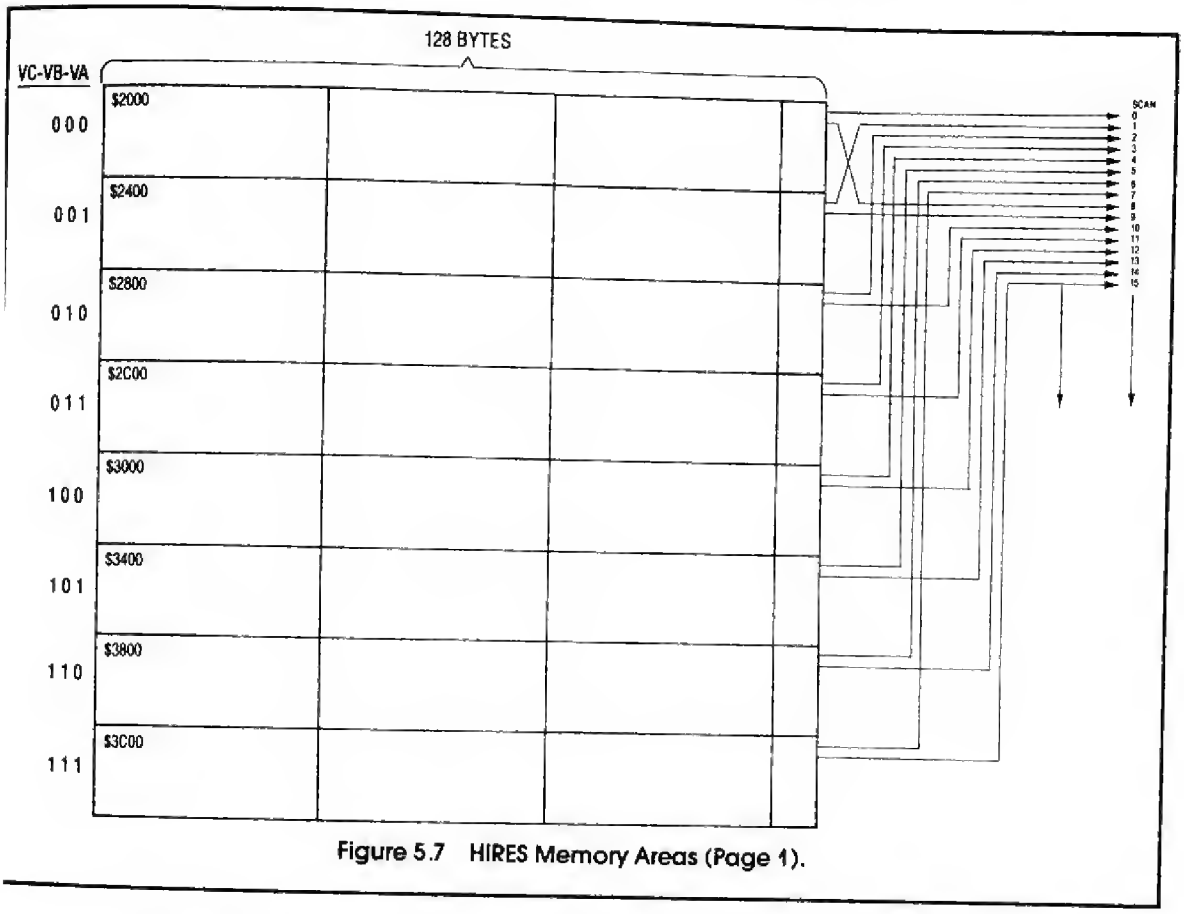


Figure 5.7 HIRES Memory Areas (Page 1).

igure gives a different perspective for viewing IIRES memory usage. Figure 5.9 is the full HIRES iemory map showing addresses scanned during IBL and VBL, as well as the displayed map. It was lso made by an Applesoft program listed in Appenix D. For reference, "#" is used in Figure 5.9 to iow when horizontal or vertical television sync is itput from the IOU. The #s in the middle of every BL period represent the horizontal sync which iuses the horizontal retrace. The long strings of #s lines 224—227 represent the vertical sync which iuses the vertical retrace (60 Hz IOU). Please note at video sync in the Apple IIe is identical to that of e RFI Revision Apple II, so the sync generation picted in Figures 5.6 and 5.9 is valid for both mputers.

The scanning during blanking periods in HIRES very similar to that in TEXT/LORES. The memy locations scanned during HBL prior to a disayed line are the 24 bytes just below the displayed ea, using 128-byte wraparound addressing. The ea scanned during VBL is the same as the top third of the screen minus eight bytes. Memory scanned by lines 256 through 261 is identical to memory scanned by lines 250 through 255, so those six 64-byte sections are scanned twice, as shown in Figure 5.9. The memory scanning areas are summarized in Table 5.2. This same information is displayed graphically in Figure 5.17.

Mixed Mode Scanning

HIRES graphics mixed with TEXT is a special case when it comes to video scanner addressing. Part of HIRES memory and part of TEXT/LORES memory must be scanned in this mode. The problem does not arise with LORES graphics mixed with TEXT, because TEXT memory scanning is identical to LORES memory scanning.

The HIRES TIME term that is used to develop the video scan address is not a direct input from the $C056/$C057 LORES/HIRES soft switch. Rather, it is a term developed in the IOU which is active when it is actually time to scan HIRES display memory. In HIRES MIXED mode, the HIRES

PAGE 1	PAGE 2	TOP SCREEN/ FIRST 40		MIDDLE SCREEN/ SECOND 40		BOTTOM SCREEN/ THIRD 40		UNUSED 8
		LIN#	PAGE 1 RANGE	LIN#	PAGE 1 RANGE	LIN#	PAGE 1 RANGE	PAGE 1 RANGE
$2000 8192	$4000 16384	000	$2000-$2027	064	$2028-$204F	128	$2050-$2077	$2078-$207F
$2400 9216	$4400 17408	001	$2400-$2427	065	$2428-$244F	129	$2450-$2477	$2478-$247F
$2800 10240	$4800 18432	002	$2800-$2827	066	$2828-$284F	130	$2850-$2877	$2878-$287F
$2C00 11264	$4C00 19456	003	$2C00-$2C27	067	$2C28-$2C4F	131	$2C50-$2C77	$2C78-$2C7F
$3000 12288	$5000 20480	004	$3000-$3027	068	$3028-$304F	132	$3050-$3077	$3078-$307F
$3400 13312	$5400 21504	005	$3400-$3427	069	$3428-$344F	133	$3450-$3477	$3478-$347F
$3800 14336	$5800 22528	006	$3800-$3827	070	$3828-$384F	134	$3850-$3877	$3878-$387F
$3C00 15360	$5C00 23552	007	$3C00-$3C27	071	$3C28-$3C4F	135	$3C50-$3C77	$3C78-$3C7F
$2080 8320	$4080 16512	008	$2080-$20A7	072	$20A8-$20CF	136	$20D0-$20F7	$20F8-$20FF
$2480 9344	$4480 17536	009	$2480-$24A7	073	$24A8-$24CF	137	$24D0-$24F7	$24F8-$24FF
$2880 10368	$4880 18560	010	$2880-$28A7	074	$28A8-$28CF	138	$28D0-$28F7	$28F8-$28FF
$2C80 11392	$4C80 19584	011	$2C80-$2CA7	075	$2CA8-$2CCF	139	$2CD0-$2CF7	$2CF8-$2CFF
$3080 12416	$5080 20608	012	$3080-$30A7	076	$30A8-$30CF	140	$30D0-$30F7	$30F8-$30FF
$3480 13440	$5480 21632	013	$3480-$34A7	077	$34A8-$34CF	141	$34D0-$34F7	$34F8-$34FF
$3880 14464	$5880 22656	014	$3880-$38A7	078	$38A8-$38CF	142	$38D0-$38F7	$38F8-$38FF
$3C80 15488	$5C80 23680	015	$3C80-$3CA7	079	$3CA8-$3CCF	143	$3CD0-$3CF7	$3CF8-$3CFF
$2100 8448	$4100 16640	016	$2100-$2127	080	$2128-$214F	144	$2150-$2177	$2178-$217F
$2500 9472	$4500 17664	017	$2500-$2527	081	$2528-$254F	145	$2550-$2577	$2578-$257F
$2900 10496	$4900 18688	018	$2900-$2927	082	$2928-$294F	146	$2950-$2977	$2978-$297F
$2D00 11520	$4D00 19712	019	$2D00-$2D27	083	$2D28-$2D4F	147	$2D50-$2D77	$2D78-$2D7F
$3100 12544	$5100 20736	020	$3100-$3127	084	$3128-$314F	148	$3150-$3177	$3178-$317F
$3500 13568	$5500 21760	021	$3500-$3527	085	$3528-$354F	149	$3550-$3577	$3578-$357F
$3900 14592	$5900 22784	022	$3900-$3927	086	$3928-$394F	150	$3950-$3977	$3978-$397F
$3D00 15616	$5D00 23808	023	$3D00-$3D27	087	$3D28-$3D4F	151	$3D50-$3D77	$3D78-$3D7F
$2180 8576	$4180 16768	024	$2180-$21A7	088	$21A8-$21CF	152	$21D0-$21F7	$21F8-$21FF
$2580 9600	$4580 17792	025	$2580-$25A7	089	$25A8-$25CF	153	$25D0-$25F7	$25F8-$25FF
$2980 10624	$4980 18816	026	$2980-$29A7	090	$29A8-$29CF	154	$29D0-$29F7	$29F8-$29FF
$2D80 11648	$4D80 19840	027	$2D80-$2DA7	091	$2DA8-$2DCF	155	$2DD0-$2DF7	$2DF8-$2DFF
$3180 12672	$5180 20864	028	$3180-$31A7	092	$31A8-$31CF	156	$31D0-$31F7	$31F8-$31FF
$3580 13696	$5580 21888	029	$3580-$35A7	093	$35A8-$35CF	157	$35D0-$35F7	$35F8-$35FF
$3980 14720	$5980 22912	030	$3980-$39A7	094	$39A8-$39CF	158	$39D0-$39F7	$39F8-$39FF
$3D80 15744	$5D80 23936	031	$3D80-$3DA7	095	$3DA8-$3DCF	159	$3DD0-$3DF7	$3DF8-$3DFF
$2200 8704	$4200 16896	032	$2200-$2227	096	$2228-$224F	160	$2250-$2277	$2278-$227F
$2600 9728	$4600 17920	033	$2600-$2627	097	$2628-$264F	161	$2650-$2677	$2678-$267F
$2A00 10752	$4A00 18944	034	$2A00-$2A27	098	$2A28-$2A4F	162	$2A50-$2A77	$2A78-$2A7F
$2E00 11776	$4E00 19968	035	$2E00-$2E27	099	$2E28-$2E4F	163	$2E50-$2E77	$2E78-$2E7F
$3200 12800	$5200 20992	036	$3200-$3227	100	$3228-$324F	164	$3250-$3277	$3278-$327F
$3600 13824	$5600 22016	037	$3600-$3627	101	$3628-$364F	165	$3650-$3677	$3678-$367F
$3A00 14848	$5A00 23040	038	$3A00-$3A27	102	$3A28-$3A4F	166	$3A50-$3A77	$3A78-$3A7F
$3E00 15872	$5E00 24064	039	$3E00-$3E27	103	$3E28-$3E4F	167	$3E50-$3E77	$3E78-$3E7F
$2280 8832	$4280 17024	040	$2280-$22A7	104	$22A8-$22CF	168	$22D0-$22F7	$22F8-$22FF
$2680 9856	$4680 18048	041	$2680-$26A7	105	$26A8-$26CF	169	$26D0-$26F7	$26F8-$26FF
$2A80 10880	$4A80 19072	042	$2A80-$2AA7	106	$2AA8-$2ACF	170	$2AD0-$2AF7	$2AF8-$2AFF
$2E80 11904	$4E80 20096	043	$2E80-$2EA7	107	$2EA8-$2ECF	171	$2ED0-$2EF7	$2EF8-$2EFF
$3280 12928	$5280 21120	044	$3280-$32A7	108	$32A8-$32CF	172	$32D0-$32F7	$32F8-$32FF
$3680 13952	$5680 22144	045	$3680-$36A7	109	$36A8-$36CF	173	$36D0-$36F7	$36F8-$36FF
$3A80 14976	$5A80 23168	046	$3A80-$3AA7	110	$3AA8-$3ACF	174	$3AD0-$3AF7	$3AF8-$3AFF
$3E80 16000	$5E80 24192	047	$3E80-$3EA7	111	$3EA8-$3ECF	175	$3ED0-$3EF7	$3EF8-$3EFF
$2300 8960	$4300 17152	048	$2300-$2327	112	$2328-$234F	176	$2350-$2377	$2378-$237F
$2700 9984	$4700 18176	049	$2700-$2727	113	$2728-$274F	177	$2750-$2777	$2778-$277F
$2B00 11008	$4B00 19200	050	$2B00-$2B27	114	$2B28-$2B4F	178	$2B50-$2B77	$2B78-$2B7F
$2F00 12032	$4F00 20224	051	$2F00-$2F27	115	$2F28-$2F4F	179	$2F50-$2F77	$2F78-$2F7F
$3300 13056	$5300 21248	052	$3300-$3327	116	$3328-$334F	180	$3350-$3377	$3378-$337F
$3700 14080	$5700 22272	053	$3700-$3727	117	$3728-$374F	181	$3750-$3777	$3778-$377F
$3B00 15104	$5B00 23296	054	$3B00-$3B27	118	$3B28-$3B4F	182	$3B50-$3B77	$3B78-$3B7F
$3F00 16128	$5F00 24320	055	$3F00-$3F27	119	$3F28-$3F4F	183	$3F50-$3F77	$3F78-$3F7F
$2380 9088	$4380 17280	056	$2380-$23A7	120	$23A8-$23CF	184	$23D0-$23F7	$23F8-$23FF
$2780 10112	$4780 18304	057	$2780-$27A7	121	$27A8-$27CF	185	$27D0-$27F7	$27F8-$27FF
$2B80 11136	$4B80 19328	058	$2B80-$2BA7	122	$2BA8-$2BCF	186	$2BD0-$2BF7	$2BF8-$2BFF
$2F80 12160	$4F80 20352	059	$2F80-$2FA7	123	$2FA8-$2FCF	187	$2FD0-$2FF7	$2FF8-$2FFF
$3380 13184	$5380 21376	060	$3380-$33A7	124	$33A8-$33CF	188	$33D0-$33F7	$33F8-$33FF
$3780 14208	$5780 22400	061	$3780-$37A7	125	$37A8-$37CF	189	$37D0-$37F7	$37F8-$37FF
$3B80 15232	$5B80 23424	062	$3B80-$3BA7	126	$3BA8-$3BCF	190	$3BD0-$3BF7	$3BF8-$3BFF
$3F80 16256	$5F80 24448	063	$3F80-$3FA7	127	$3FA8-$3FCF	191	$3FD0-$3FF7	$3FF8-$3FFF

Figure 5.8 HIRES Displayed Memory Map.

	SCREEN TOP					
	HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
LINE NUM	PAGE 1	PAGE 2	11111111 Ø0123456789ABCDEFØ1234567	PAGE 1	PAGE 2	1111111111111111122222222 0123456789ABCDEFØ123456789ABCDEFØ1234567
Ø	$2Ø68 8296	$4Ø68 16488	++++++++++####++++++++++	$2Ø00 8192	$4Ø00 16384	++++++++++++++++++++++++++++++++
1	$2468 9320	$4468 17512	++++++++++####++++++++++	$2400 9216	$4400 17408	++++++++++++++++++++++++++++++++
2	$2868 10344	$4868 18536	++++++++++####++++++++++	$2800 10240	$4800 18432	++++++++++++++++++++++++++++++++
3	$2C68 11368	$4C68 19560	++++++++++####++++++++++	$2C00 11264	$4C00 19456	++++++++++++++++++++++++++++++++
4	$3068 12392	$5068 20584	++++++++++####++++++++++	$3000 12288	$5000 20480	++++++++++++++++++++++++++++++++
5	$3468 13416	$5468 21608	++++++++++####++++++++++	$3400 13312	$5400 21504	++++++++++++++++++++++++++++++++
6	$3868 14440	$5868 22632	++++++++++####++++++++++	$3800 14336	$5800 22528	++++++++++++++++++++++++++++++++
7	$3C68 15464	$5C68 23656	++++++++++####++++++++++	$3C00 15360	$5C00 23552	++++++++++++++++++++++++++++++++
8	$20E8 8424	$40E8 16616	++++++++++####++++++++++	$2080 8320	$4080 16512	++++++++++++++++++++++++++++++++
9	$24E8 9448	$44E8 17640	++++++++++####++++++++++	$2480 9344	$4480 17536	++++++++++++++++++++++++++++++++
10	$28E8 10472	$48E8 18664	++++++++++####++++++++++	$2880 10368	$4880 18560	++++++++++++++++++++++++++++++++
11	$2CE8 11496	$4CE8 19688	++++++++++####++++++++++	$2C80 11392	$4C80 19584	++++++++++++++++++++++++++++++++
12	$30E8 12520	$50E8 20712	++++++++++####++++++++++	$3080 12416	$5080 20608	++++++++++++++++++++++++++++++++
13	$34E8 13544	$54E8 21736	++++++++++####++++++++++	$3480 13440	$5480 21632	++++++++++++++++++++++++++++++++
14	$38E8 14568	$58E8 22760	++++++++++####++++++++++	$3880 14464	$5880 22656	++++++++++++++++++++++++++++++++
15	$3CE8 15592	$5CE8 23784	++++++++++####++++++++++	$3C80 15488	$5C80 23680	++++++++++++++++++++++++++++++++
16	$2168 8552	$4168 16744	++++++++++####++++++++++	$2100 8448	$4100 16640	++++++++++++++++++++++++++++++++
17	$2568 9576	$4568 17768	++++++++++####++++++++++	$2500 9472	$4500 17664	++++++++++++++++++++++++++++++++
18	$2968 10600	$4968 18792	++++++++++####++++++++++	$2900 10496	$4900 18688	++++++++++++++++++++++++++++++++
19	$2D68 11624	$4D68 19816	++++++++++####++++++++++	$2D00 11520	$4D00 19712	++++++++++++++++++++++++++++++++
20	$3168 12648	$5168 20840	++++++++++####++++++++++	$3100 12544	$5100 20736	++++++++++++++++++++++++++++++++
21	$3568 13672	$5568 21864	++++++++++####++++++++++	$3500 13568	$5500 21760	++++++++++++++++++++++++++++++++
22	$3968 14696	$5968 22888	++++++++++####++++++++++	$3900 14592	$5900 22784	++++++++++++++++++++++++++++++++
23	$3D68 15720	$5D68 23912	++++++++++####++++++++++	$3D00 15616	$5D00 23808	++++++++++++++++++++++++++++++++
24	$21E8 8680	$41E8 16872	++++++++++####++++++++++	$2180 8576	$4180 16768	++++++++++++++++++++++++++++++++
25	$25E8 9704	$45E8 17896	++++++++++####++++++++++	$2580 9600	$4580 17792	++++++++++++++++++++++++++++++++
26	$29E8 10728	$49E8 18920	++++++++++####++++++++++	$2980 10624	$4980 18816	++++++++++++++++++++++++++++++++
27	$2DE8 11752	$4DE8 19944	++++++++++####++++++++++	$2D80 11648	$4D80 19840	++++++++++++++++++++++++++++++++
28	$31E8 12776	$51E8 20968	++++++++++####++++++++++	$3180 12672	$5180 20864	++++++++++++++++++++++++++++++++
29	$35E8 13800	$55E8 21992	++++++++++####++++++++++	$3580 13696	$5580 21888	++++++++++++++++++++++++++++++++
30	$39E8 14824	$59E8 23016	++++++++++####++++++++++	$3980 14720	$5980 22912	++++++++++++++++++++++++++++++++
31	$3DE8 15848	$5DE8 24040	++++++++++####++++++++++	$3D80 15744	$5D80 23936	++++++++++++++++++++++++++++++++
32	$2268 8808	$4268 17000	++++++++++####++++++++++	$2200 8704	$4200 16896	++++++++++++++++++++++++++++++++
33	$2668 9832	$4668 18024	++++++++++####++++++++++	$2600 9728	$4600 17920	++++++++++++++++++++++++++++++++
34	$2A68 10856	$4A68 19048	++++++++++####++++++++++	$2A00 10752	$4A00 18944	++++++++++++++++++++++++++++++++
35	$2E68 11880	$4E68 20072	++++++++++####++++++++++	$2E00 11776	$4E00 19968	++++++++++++++++++++++++++++++++
36	$3268 12904	$5268 21096	++++++++++####++++++++++	$3200 12800	$5200 20992	++++++++++++++++++++++++++++++++
37	$3668 13928	$5668 22120	++++++++++####++++++++++	$3600 13824	$5600 22016	++++++++++++++++++++++++++++++++
38	$3A68 14952	$5A68 23144	++++++++++####++++++++++	$3A00 14848	$5A00 23040	++++++++++++++++++++++++++++++++
39	$3E68 15976	$5E68 24168	++++++++++####++++++++++	$3E00 15872	$5E00 24064	++++++++++++++++++++++++++++++++
40	$22E8 8936	$42E8 17128	++++++++++####++++++++++	$2280 8832	$4280 17024	++++++++++++++++++++++++++++++++
41	$26E8 9960	$46E8 18152	++++++++++####++++++++++	$2680 9856	$4680 18048	++++++++++++++++++++++++++++++++
42	$2AE8 10984	$4AE8 19176	++++++++++####++++++++++	$2A80 10880	$4A80 19072	++++++++++++++++++++++++++++++++
43	$2EE8 12008	$4EE8 20200	++++++++++####++++++++++	$2E80 11904	$4E80 20096	++++++++++++++++++++++++++++++++
44	$32E8 13032	$52E8 21224	++++++++++####++++++++++	$3280 12928	$5280 21120	++++++++++++++++++++++++++++++++
45	$36E8 14056	$56E8 22248	++++++++++####++++++++++	$3680 13952	$5680 22144	++++++++++++++++++++++++++++++++
46	$3AE8 15080	$5AE8 23272	++++++++++####++++++++++	$3A80 14976	$5A80 23168	++++++++++++++++++++++++++++++++
47	$3EE8 16104	$5EE8 24296	++++++++++####++++++++++	$3E80 16000	$5E80 24192	++++++++++++++++++++++++++++++++
48	$2368 9064	$4368 17256	++++++++++####++++++++++	$2300 8960	$4300 17152	++++++++++++++++++++++++++++++++
49	$2768 10088	$4768 18280	++++++++++####++++++++++	$2700 9984	$4700 18176	++++++++++++++++++++++++++++++++
50	$2B68 11112	$4B68 19304	++++++++++####++++++++++	$2B00 11008	$4B00 19200	++++++++++++++++++++++++++++++++
51	$2F68 12136	$4F68 20328	++++++++++####++++++++++	$2F00 12032	$4F00 20224	++++++++++++++++++++++++++++++++
52	$3368 13160	$5368 21352	++++++++++####++++++++++	$3300 13056	$5300 21248	++++++++++++++++++++++++++++++++
53	$3768 14184	$5768 22376	++++++++++####++++++++++	$3700 14080	$5700 22272	++++++++++++++++++++++++++++++++
54	$3B68 15208	$5B68 23400	++++++++++####++++++++++	$3B00 15104	$5B00 23296	++++++++++++++++++++++++++++++++
55	$3F68 16232	$5F68 24424	++++++++++####++++++++++	$3F00 16128	$5F00 24320	++++++++++++++++++++++++++++++++
56	$23E8 9192	$43E8 17384	++++++++++####++++++++++	$2380 9088	$4380 17280	++++++++++++++++++++++++++++++++
57	$27E8 10216	$47E8 18408	++++++++++####++++++++++	$2780 10112	$4780 18304	++++++++++++++++++++++++++++++++
58	$2BE8 11240	$4BE8 19432	++++++++++####++++++++++	$2B80 11136	$4B80 19328	++++++++++++++++++++++++++++++++
59	$2FE8 12264	$4FE8 20456	++++++++++####++++++++++	$2F80 12160	$4F80 20352	++++++++++++++++++++++++++++++++
60	$33E8 13288	$53E8 21480	++++++++++####++++++++++	$3380 13184	$5380 21376	++++++++++++++++++++++++++++++++
61	$37E8 14312	$57E8 22504	++++++++++####++++++++++	$3780 14208	$5780 22400	++++++++++++++++++++++++++++++++
62	$3BE8 15336	$5BE8 23528	++++++++++####++++++++++	$3B80 15232	$5B80 23424	++++++++++++++++++++++++++++++++
63	$3FE8 16360	$5FE8 24552	++++++++++####++++++++++	$3F80 16256	$5F80 24448	++++++++++++++++++++++++++++++++

Figure 5.9 HIRES Scanning Map Including Undisplayed Areas (1 of 4).

	SCREEN MIDDLE					
	HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
LINE NUM	PAGE 1	PAGE 2	11111111 0012345678 9ABCDEF01234567	PAGE 1	PAGE 2	1111111111111112222222 0123456789ABCDEF0123456789ABCDEF01234567
64	$2010 8208	$4010 16400	++++++++++####++++++++++	$2028 8232	$4028 16424	++++++++++++++++++++++++++++++++
65	$2410 9232	$4410 17424	++++++++++####++++++++++	$2428 9256	$4428 17448	++++++++++++++++++++++++++++++++
66	$2810 10256	$4810 18448	++++++++++####++++++++++	$2828 10280	$4828 18472	++++++++++++++++++++++++++++++++
67	$2C10 11280	$4C10 19472	++++++++++####++++++++++	$2C28 11304	$4C28 19496	++++++++++++++++++++++++++++++++
68	$3010 12304	$5010 20496	++++++++++####++++++++++	$3028 12328	$5028 20520	++++++++++++++++++++++++++++++++
69	$3410 13328	$5410 21520	++++++++++####++++++++++	$3428 13352	$5428 21544	++++++++++++++++++++++++++++++++
70	$3810 14352	$5810 22544	++++++++++####++++++++++	$3828 14376	$5828 22568	++++++++++++++++++++++++++++++++
71	$3C10 15376	$5C10 23568	++++++++++####++++++++++	$3C28 15400	$5C28 23592	++++++++++++++++++++++++++++++++
72	$2090 8336	$4090 16528	++++++++++####++++++++++	$20A8 8360	$40A8 16552	++++++++++++++++++++++++++++++++
73	$2490 9360	$4490 17552	++++++++++####++++++++++	$24A8 9384	$44A8 17576	++++++++++++++++++++++++++++++++
74	$2890 10384	$4890 18576	++++++++++####++++++++++	$28A8 10408	$48A8 18600	++++++++++++++++++++++++++++++++
75	$2C90 11408	$4C90 19600	++++++++++####++++++++++	$2CA8 11432	$4CA8 19624	++++++++++++++++++++++++++++++++
76	$3090 12432	$5090 20624	++++++++++####++++++++++	$30A8 12456	$50A8 20648	++++++++++++++++++++++++++++++++
77	$3490 13456	$5490 21648	++++++++++####++++++++++	$34A8 13480	$54A8 21672	++++++++++++++++++++++++++++++++
78	$3890 14480	$5890 22672	++++++++++####++++++++++	$38A8 14504	$58A8 22696	++++++++++++++++++++++++++++++++
79	$3C90 15504	$5C90 23696	++++++++++####++++++++++	$3CA8 15528	$5CA8 23720	++++++++++++++++++++++++++++++++
80	$2110 8464	$4110 16656	++++++++++####++++++++++	$2128 8488	$4128 16680	++++++++++++++++++++++++++++++++
81	$2510 9488	$4510 17680	++++++++++####++++++++++	$2528 9512	$4528 17704	++++++++++++++++++++++++++++++++
82	$2910 10512	$4910 18704	++++++++++####++++++++++	$2928 10536	$4928 18728	++++++++++++++++++++++++++++++++
83	$2D10 11536	$4D10 19728	++++++++++####++++++++++	$2D28 11560	$4D28 19752	++++++++++++++++++++++++++++++++
84	$3110 12560	$5110 20752	++++++++++####++++++++++	$3128 12584	$5128 20776	++++++++++++++++++++++++++++++++
85	$3510 13584	$5510 21776	++++++++++####++++++++++	$3528 13608	$5528 21800	++++++++++++++++++++++++++++++++
86	$3910 14608	$5910 22800	++++++++++####++++++++++	$3928 14632	$5928 22824	++++++++++++++++++++++++++++++++
87	$3D10 15632	$5D10 23824	++++++++++####++++++++++	$3D28 15656	$5D28 23848	++++++++++++++++++++++++++++++++
88	$2190 8592	$4190 16784	++++++++++####++++++++++	$21A8 8616	$41A8 16808	++++++++++++++++++++++++++++++++
89	$2590 9616	$4590 17808	++++++++++####++++++++++	$25A8 9640	$45A8 17832	++++++++++++++++++++++++++++++++
90	$2990 10640	$4990 18832	++++++++++####++++++++++	$29A8 10664	$49A8 18856	++++++++++++++++++++++++++++++++
91	$2D90 11664	$4D90 19856	++++++++++####++++++++++	$2DA8 11688	$4DA8 19880	++++++++++++++++++++++++++++++++
92	$3190 12688	$5190 20880	++++++++++####++++++++++	$31A8 12712	$51A8 20904	++++++++++++++++++++++++++++++++
93	$3590 13712	$5590 21904	++++++++++####++++++++++	$35A8 13736	$55A8 21928	++++++++++++++++++++++++++++++++
94	$3990 14736	$5990 22928	++++++++++####++++++++++	$39A8 14760	$59A8 22952	++++++++++++++++++++++++++++++++
95	$3D90 15760	$5D90 23952	++++++++++####++++++++++	$3DA8 15784	$5DA8 23976	++++++++++++++++++++++++++++++++
96	$2210 8720	$4210 16912	++++++++++####++++++++++	$2228 8744	$4228 16936	++++++++++++++++++++++++++++++++
97	$2610 9744	$4610 17936	++++++++++####++++++++++	$2628 9768	$4628 17960	++++++++++++++++++++++++++++++++
98	$2A10 10768	$4A10 18960	++++++++++####++++++++++	$2A28 10792	$4A28 18984	++++++++++++++++++++++++++++++++
99	$2E10 11792	$4E10 19984	++++++++++####++++++++++	$2E28 11816	$4E28 20008	++++++++++++++++++++++++++++++++
100	$3210 12816	$5210 21008	++++++++++####++++++++++	$3228 12840	$5228 21032	++++++++++++++++++++++++++++++++
101	$3610 13840	$5610 22032	++++++++++####++++++++++	$3628 13864	$5628 22056	++++++++++++++++++++++++++++++++
102	$3A10 14864	$5A10 23056	++++++++++####++++++++++	$3A28 14888	$5A28 23080	++++++++++++++++++++++++++++++++
103	$3E10 15888	$5E10 24080	++++++++++####++++++++++	$3E28 15912	$5E28 24104	++++++++++++++++++++++++++++++++
104	$2290 8848	$4290 17040	++++++++++####++++++++++	$22A8 8872	$42A8 17064	++++++++++++++++++++++++++++++++
105	$2690 9872	$4690 18064	++++++++++####++++++++++	$26A8 9896	$46A8 18088	++++++++++++++++++++++++++++++++
106	$2A90 10896	$4A90 19088	++++++++++####++++++++++	$2AA8 10920	$4AA8 19112	++++++++++++++++++++++++++++++++
107	$2E90 11920	$4E90 20112	++++++++++####++++++++++	$2EA8 11944	$4EA8 20136	++++++++++++++++++++++++++++++++
108	$3290 12944	$5290 21136	++++++++++####++++++++++	$32A8 12968	$52A8 21160	++++++++++++++++++++++++++++++++
109	$3690 13968	$5690 22160	++++++++++####++++++++++	$36A8 13992	$56A8 22184	++++++++++++++++++++++++++++++++
110	$3A90 14992	$5A90 23184	++++++++++####++++++++++	$3AA8 15016	$5AA8 23208	++++++++++++++++++++++++++++++++
111	$3E90 16016	$5E90 24208	++++++++++####++++++++++	$3EA8 16040	$5EA8 24232	++++++++++++++++++++++++++++++++
112	$2310 8976	$4310 17168	++++++++++####++++++++++	$2328 9000	$4328 17192	++++++++++++++++++++++++++++++++
113	$2710 10000	$4710 18192	++++++++++####++++++++++	$2728 10024	$4728 18216	++++++++++++++++++++++++++++++++
114	$2B10 11024	$4B10 19216	++++++++++####++++++++++	$2B28 11048	$4B28 19240	++++++++++++++++++++++++++++++++
115	$2F10 12048	$4F10 20240	++++++++++####++++++++++	$2F28 12072	$4F28 20264	++++++++++++++++++++++++++++++++
116	$3310 13072	$5310 21264	++++++++++####++++++++++	$3328 13096	$5328 21288	++++++++++++++++++++++++++++++++
117	$3710 14096	$5710 22288	++++++++++####++++++++++	$3728 14120	$5728 22312	++++++++++++++++++++++++++++++++
118	$3B10 15120	$5B10 23312	++++++++++####++++++++++	$3B28 15144	$5B28 23336	++++++++++++++++++++++++++++++++
119	$3F10 16144	$5F10 24336	++++++++++####++++++++++	$3F28 16168	$5F28 24360	++++++++++++++++++++++++++++++++
120	$2390 9104	$4390 17296	++++++++++####++++++++++	$23A8 9128	$43A8 17320	++++++++++++++++++++++++++++++++
121	$2790 10128	$4790 18320	++++++++++####++++++++++	$27A8 10152	$47A8 18344	++++++++++++++++++++++++++++++++
122	$2B90 11152	$4B90 19344	++++++++++####++++++++++	$2BA8 11176	$4BA8 19368	++++++++++++++++++++++++++++++++
123	$2F90 12176	$4F90 20368	++++++++++####++++++++++	$2FA8 12200	$4FA8 20392	++++++++++++++++++++++++++++++++
124	$3390 13200	$5390 21392	++++++++++####++++++++++	$33A8 13224	$53A8 21416	++++++++++++++++++++++++++++++++
125	$3790 14224	$5790 22416	++++++++++####++++++++++	$37A8 14248	$57A8 22440	++++++++++++++++++++++++++++++++
126	$3B90 15248	$5B90 23440	++++++++++####++++++++++	$3BA8 15272	$5BA8 23464	++++++++++++++++++++++++++++++++
127	$3F90 16272	$5F90 24464	++++++++++####++++++++++	$3FA8 16296	$5FA8 24488	++++++++++++++++++++++++++++++++

Figure 5.9 HIRES Scanning Map Including Undisplayed Areas (2 of 4).

	SCREEN BOTTOM					
	HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
LINE NUM	PAGE 1	PAGE 2	11111111 Ø0123456789ABCDEFØ1234567	PAGE 1	PAGE 2	1111111111111111222222222 Ø123456789ABCDEFØ123456789ABCDEFØ1234567
128	$2038 8248	$4038 16440	++++++++####++++++++	$2050 8272	$4050 16464	++++++++++++++++++++++++++++++++
129	$2438 9272	$4438 17464	++++++++####++++++++	$2450 9296	$4450 17488	++++++++++++++++++++++++++++++++
130	$2838 10296	$4838 18488	++++++++####++++++++	$2850 10320	$4850 18512	++++++++++++++++++++++++++++++++
131	$2C38 11320	$4C38 19512	++++++++####++++++++	$2C50 11344	$4C50 19536	++++++++++++++++++++++++++++++++
132	$3038 12344	$5038 20536	++++++++####++++++++	$3050 12368	$5050 20560	++++++++++++++++++++++++++++++++
133	$3438 13368	$5438 21560	++++++++####++++++++	$3450 13392	$5450 21584	++++++++++++++++++++++++++++++++
134	$3838 14392	$5838 22584	++++++++####++++++++	$3850 14416	$5850 22608	++++++++++++++++++++++++++++++++
135	$3C38 15416	$5C38 23608	++++++++####++++++++	$3C50 15440	$5C50 23632	++++++++++++++++++++++++++++++++
136	$20B8 8376	$40B8 16568	++++++++####++++++++	$20D0 8400	$40D0 16592	++++++++++++++++++++++++++++++++
137	$24B8 9400	$44B8 17592	++++++++####++++++++	$24D0 9424	$44D0 17616	++++++++++++++++++++++++++++++++
138	$28B8 10424	$48B8 18616	++++++++####++++++++	$28D0 10448	$48D0 18640	++++++++++++++++++++++++++++++++
139	$2CB8 11448	$4CB8 19640	++++++++####++++++++	$2CD0 11472	$4CD0 19664	++++++++++++++++++++++++++++++++
140	$30B8 12472	$50B8 20664	++++++++####++++++++	$30D0 12496	$50D0 20688	++++++++++++++++++++++++++++++++
141	$34B8 13496	$54B8 21688	++++++++####++++++++	$34D0 13520	$54D0 21712	++++++++++++++++++++++++++++++++
142	$38B8 14520	$58B8 22712	++++++++####++++++++	$38D0 14544	$58D0 22736	++++++++++++++++++++++++++++++++
143	$3CB8 15544	$5CB8 23736	++++++++####++++++++	$3CD0 15568	$5CD0 23760	++++++++++++++++++++++++++++++++
144	$2138 8504	$4138 16696	++++++++####++++++++	$2150 8528	$4150 16720	++++++++++++++++++++++++++++++++
145	$2538 9528	$4538 17720	++++++++####++++++++	$2550 9552	$4550 17744	++++++++++++++++++++++++++++++++
146	$2938 10552	$4938 18744	++++++++####++++++++	$2950 10576	$4950 18768	++++++++++++++++++++++++++++++++
147	$2D38 11576	$4D38 19768	++++++++####++++++++	$2D50 11600	$4D50 19792	++++++++++++++++++++++++++++++++
148	$3138 12600	$5138 20792	++++++++####++++++++	$3150 12624	$5150 20816	++++++++++++++++++++++++++++++++
149	$3538 13624	$5538 21816	++++++++####++++++++	$3550 13648	$5550 21840	++++++++++++++++++++++++++++++++
150	$3938 14648	$5938 22840	++++++++####++++++++	$3950 14672	$5950 22864	++++++++++++++++++++++++++++++++
151	$3D38 15672	$5D38 23864	++++++++####++++++++	$3D50 15696	$5D50 23888	++++++++++++++++++++++++++++++++
152	$21B8 8632	$41B8 16824	++++++++####++++++++	$21D0 8656	$41D0 16848	++++++++++++++++++++++++++++++++
153	$25B8 9656	$45B8 17848	++++++++####++++++++	$25D0 9680	$45D0 17872	++++++++++++++++++++++++++++++++
154	$29B8 10680	$49B8 18872	++++++++####++++++++	$29D0 10704	$49D0 18896	++++++++++++++++++++++++++++++++
155	$2DB8 11704	$4DB8 19896	++++++++####++++++++	$2DD0 11728	$4DD0 19920	++++++++++++++++++++++++++++++++
156	$31B8 12728	$51B8 20920	++++++++####++++++++	$31D0 12752	$51D0 20944	++++++++++++++++++++++++++++++++
157	$35B8 13752	$55B8 21944	++++++++####++++++++	$35D0 13776	$55D0 21968	++++++++++++++++++++++++++++++++
158	$39B8 14776	$59B8 22968	++++++++####++++++++	$39D0 14800	$59D0 22992	++++++++++++++++++++++++++++++++
159	$3DB8 15800	$5DB8 23992	++++++++####++++++++	$3DD0 15824	$5DD0 24016	++++++++++++++++++++++++++++++++
160	$2238 8760	$4238 16952	++++++++####++++++++	$2250 8784	$4250 16976	++++++++++++++++++++++++++++++++
161	$2638 9784	$4638 17976	++++++++####++++++++	$2650 9808	$4650 18000	++++++++++++++++++++++++++++++++
162	$2A38 10808	$4A38 19000	++++++++####++++++++	$2A50 10832	$4A50 19024	++++++++++++++++++++++++++++++++
163	$2E38 11832	$4E38 20024	++++++++####++++++++	$2E50 11856	$4E50 20048	++++++++++++++++++++++++++++++++
164	$3238 12856	$5238 21048	++++++++####++++++++	$3250 12880	$5250 21072	++++++++++++++++++++++++++++++++
165	$3638 13880	$5638 22072	++++++++####++++++++	$3650 13904	$5650 22096	++++++++++++++++++++++++++++++++
166	$3A38 14904	$5A38 23096	++++++++####++++++++	$3A50 14928	$5A50 23120	++++++++++++++++++++++++++++++++
167	$3E38 15928	$5E38 24120	++++++++####++++++++	$3E50 15952	$5E50 24144	++++++++++++++++++++++++++++++++
168	$22B8 8888	$42B8 17080	++++++++####++++++++	$22D0 8912	$42D0 17104	++++++++++++++++++++++++++++++++
169	$26B8 9912	$46B8 18104	++++++++####++++++++	$26D0 9936	$46D0 18128	++++++++++++++++++++++++++++++++
170	$2AB8 10936	$4AB8 19128	++++++++####++++++++	$2AD0 10960	$4AD0 19152	++++++++++++++++++++++++++++++++
171	$2EB8 11960	$4EB8 20152	++++++++####++++++++	$2ED0 11984	$4ED0 20176	++++++++++++++++++++++++++++++++
172	$32B8 12984	$52B8 21176	++++++++####++++++++	$32D0 13008	$52D0 21200	++++++++++++++++++++++++++++++++
173	$36B8 14008	$56B8 22200	++++++++####++++++++	$36D0 14032	$56D0 22224	++++++++++++++++++++++++++++++++
174	$3AB8 15032	$5AB8 23224	++++++++####++++++++	$3AD0 15056	$5AD0 23248	++++++++++++++++++++++++++++++++
175	$3EB8 16056	$5EB8 24248	++++++++####++++++++	$3ED0 16080	$5ED0 24272	++++++++++++++++++++++++++++++++
176	$2338 9016	$4338 17200	++++++++####++++++++	$2350 9040	$4350 17232	++++++++++++++++++++++++++++++++
177	$2738 10040	$4738 18232	++++++++####++++++++	$2750 10064	$4750 18256	++++++++++++++++++++++++++++++++
178	$2B38 11064	$4B38 19256	++++++++####++++++++	$2B50 11088	$4B50 19280	++++++++++++++++++++++++++++++++
179	$2F38 12088	$4F38 20280	++++++++####++++++++	$2F50 12112	$4F50 20304	++++++++++++++++++++++++++++++++
180	$3338 13112	$5338 21304	++++++++####++++++++	$3350 13136	$5350 21328	++++++++++++++++++++++++++++++++
181	$3738 14136	$5738 22328	++++++++####++++++++	$3750 14160	$5750 22352	++++++++++++++++++++++++++++++++
182	$3B38 15160	$5B38 23352	++++++++####++++++++	$3B50 15184	$5B50 23376	++++++++++++++++++++++++++++++++
183	$3F38 16184	$5F38 24376	++++++++####++++++++	$3F50 16208	$5F50 24400	++++++++++++++++++++++++++++++++
184	$23B8 9144	$43B8 17336	++++++++####++++++++	$23D0 9168	$43D0 17360	++++++++++++++++++++++++++++++++
185	$27B8 10168	$47B8 18360	++++++++####++++++++	$27D0 10192	$47D0 18384	++++++++++++++++++++++++++++++++
186	$2BB8 11192	$4BB8 19384	++++++++####++++++++	$2BD0 11216	$4BD0 19408	++++++++++++++++++++++++++++++++
187	$2FB8 12216	$4FB8 20408	++++++++####++++++++	$2FD0 12240	$4FD0 20432	++++++++++++++++++++++++++++++++
188	$33B8 13240	$53B8 21432	++++++++####++++++++	$33D0 13264	$53D0 21456	++++++++++++++++++++++++++++++++
189	$37B8 14264	$57B8 22456	++++++++####++++++++	$37D0 14288	$57D0 22480	++++++++++++++++++++++++++++++++
190	$3BB8 15288	$5BB8 23480	++++++++####++++++++	$3BD0 15312	$5BD0 23504	++++++++++++++++++++++++++++++++
191	$3FB8 16312	$5FB8 24504	++++++++####++++++++	$3FD0 16336	$5FD0 24528	++++++++++++++++++++++++++++++++

Figure 5.9 HIRES Scanning Map Including Undisplayed Areas (3 of 4).

	VERTICAL BLANKING PERIOD (VBL)					
	HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
LINE NUM	PAGE 1	PAGE 2	11111111 Ø0123456789ABCDEFØ1234567	PAGE 1	PAGE 2	1111111111111111122222222 Ø123456789ABCDEFØ123456789ABCDEFØ1234567
192	$2060 8288	$4060 16480	++++++++####++++++++++	$2078 8312	$4078 16504	++++++++++++++++++++++++++++++++
193	$2460 9312	$4460 17504	++++++++####++++++++++	$2478 9336	$4478 17528	++++++++++++++++++++++++++++++++
194	$2860 10336	$4860 18528	++++++++####++++++++++	$2878 10360	$4878 18552	++++++++++++++++++++++++++++++++
195	$2C60 11360	$4C60 19552	++++++++####++++++++++	$2C78 11384	$4C78 19576	++++++++++++++++++++++++++++++++
196	$3060 12384	$5060 20576	++++++++####++++++++++	$3078 12408	$5078 20600	++++++++++++++++++++++++++++++++
197	$3460 13408	$5460 21600	++++++++####++++++++++	$3478 13432	$5478 21624	++++++++++++++++++++++++++++++++
198	$3860 14432	$5860 22624	++++++++####++++++++++	$3878 14456	$5878 22648	++++++++++++++++++++++++++++++++
199	$3C60 15456	$5C60 23648	++++++++####++++++++++	$3C78 15480	$5C78 23672	++++++++++++++++++++++++++++++++
200	$20E0 8416	$40E0 16608	++++++++####++++++++++	$20F8 8440	$40F8 16632	++++++++++++++++++++++++++++++++
201	$24E0 9440	$44E0 17632	++++++++####++++++++++	$24F8 9464	$44F8 17656	++++++++++++++++++++++++++++++++
202	$28E0 10464	$48E0 18656	++++++++####++++++++++	$28F8 10488	$48F8 18680	++++++++++++++++++++++++++++++++
203	$2CE0 11488	$4CE0 19680	++++++++####++++++++++	$2CF8 11512	$4CF8 19704	++++++++++++++++++++++++++++++++
204	$30E0 12512	$50E0 20704	++++++++####++++++++++	$30F8 12536	$50F8 20728	++++++++++++++++++++++++++++++++
205	$34E0 13536	$54E0 21728	++++++++####++++++++++	$34F8 13560	$54F8 21752	++++++++++++++++++++++++++++++++
206	$38E0 14560	$58E0 22752	++++++++####++++++++++	$38F8 14584	$58F8 22776	++++++++++++++++++++++++++++++++
207	$3CE0 15584	$5CE0 23776	++++++++####++++++++++	$3CF8 15608	$5CF8 23800	++++++++++++++++++++++++++++++++
208	$2160 8544	$4160 16736	++++++++####++++++++++	$2178 8568	$4178 16760	++++++++++++++++++++++++++++++++
209	$2560 9568	$4560 17760	++++++++####++++++++++	$2578 9592	$4578 17784	++++++++++++++++++++++++++++++++
210	$2960 10592	$4960 18784	++++++++####++++++++++	$2978 10616	$4978 18808	++++++++++++++++++++++++++++++++
211	$2D60 11616	$4D60 19808	++++++++####++++++++++	$2D78 11640	$4D78 19832	++++++++++++++++++++++++++++++++
212	$3160 12640	$5160 20832	++++++++####++++++++++	$3178 12664	$5178 20856	++++++++++++++++++++++++++++++++
213	$3560 13664	$5560 21856	++++++++####++++++++++	$3578 13688	$5578 21880	++++++++++++++++++++++++++++++++
214	$3960 14688	$5960 22880	++++++++####++++++++++	$3978 14712	$5978 22904	++++++++++++++++++++++++++++++++
215	$3D60 15712	$5D60 23904	++++++++####++++++++++	$3D78 15736	$5D78 23928	++++++++++++++++++++++++++++++++
216	$21E0 8672	$41E0 16864	++++++++####++++++++++	$21F8 8696	$41F8 16888	++++++++++++++++++++++++++++++++
217	$25E0 9696	$45E0 17888	++++++++####++++++++++	$25F8 9720	$45F8 17912	++++++++++++++++++++++++++++++++
218	$29E0 10720	$49E0 18912	++++++++####++++++++++	$29F8 10744	$49F8 18936	++++++++++++++++++++++++++++++++
219	$2DE0 11744	$4DE0 19936	++++++++####++++++++++	$2DF8 11768	$4DF8 19960	++++++++++++++++++++++++++++++++
220	$31E0 12768	$51E0 20960	++++++++####++++++++++	$31F8 12792	$51F8 20984	++++++++++++++++++++++++++++++++
221	$35E0 13792	$55E0 21984	++++++++####++++++++++	$35F8 13816	$55F8 22008	++++++++++++++++++++++++++++++++
222	$39E0 14816	$59E0 23008	++++++++####++++++++++	$39F8 14840	$59F8 23032	++++++++++++++++++++++++++++++++
223	$3DE0 15840	$5DE0 24032	++++++++####++++++++++	$3DF8 15864	$5DF8 24056	++++++++++++++++++++++++++++++++
224	$2260 8800	$4260 16992	++++++++################	$2278 8824	$4278 17016	################################
225	$2660 9824	$4660 18016	++++++++################	$2678 9848	$4678 18040	################################
226	$2A60 10848	$4A60 19040	++++++++################	$2A78 10872	$4A78 19064	################################
227	$2E60 11872	$4E60 20064	++++++++################	$2E78 11896	$4E78 20088	################################
228	$3260 12896	$5260 21088	++++++++####++++++++++	$3278 12920	$5278 21112	++++++++++++++++++++++++++++++++
229	$3660 13920	$5660 22112	++++++++####++++++++++	$3678 13944	$5678 22136	++++++++++++++++++++++++++++++++
230	$3A60 14944	$5A60 23136	++++++++####++++++++++	$3A78 14968	$5A78 23160	++++++++++++++++++++++++++++++++
231	$3E60 15968	$5E60 24160	++++++++####++++++++++	$3E78 15992	$5E78 24184	++++++++++++++++++++++++++++++++
232	$22E0 8928	$42E0 17120	++++++++####++++++++++	$22F8 8952	$42F8 17144	++++++++++++++++++++++++++++++++
233	$26E0 9952	$46E0 18144	++++++++####++++++++++	$26F8 9976	$46F8 18168	++++++++++++++++++++++++++++++++
234	$2AE0 10976	$4AE0 19168	++++++++####++++++++++	$2AF8 11000	$4AF8 19192	++++++++++++++++++++++++++++++++
235	$2EE0 12000	$4EE0 20192	++++++++####++++++++++	$2EF8 12024	$4EF8 20216	++++++++++++++++++++++++++++++++
236	$32E0 13024	$52E0 21216	++++++++####++++++++++	$32F8 13048	$52F8 21240	++++++++++++++++++++++++++++++++
237	$36E0 14048	$56E0 22240	++++++++####++++++++++	$36F8 14072	$56F8 22264	++++++++++++++++++++++++++++++++
238	$3AE0 15072	$5AE0 23264	++++++++####++++++++++	$3AF8 15096	$5AF8 23288	++++++++++++++++++++++++++++++++
239	$3EE0 16096	$5EE0 24288	++++++++####++++++++++	$3EF8 16120	$5EF8 24312	++++++++++++++++++++++++++++++++
240	$2360 9056	$4360 17248	++++++++####++++++++++	$2378 9080	$4378 17272	++++++++++++++++++++++++++++++++
241	$2760 10080	$4760 18272	++++++++####++++++++++	$2778 10104	$4778 18296	++++++++++++++++++++++++++++++++
242	$2B60 11104	$4B60 19296	++++++++####++++++++++	$2B78 11128	$4B78 19320	++++++++++++++++++++++++++++++++
243	$2F60 12128	$4F60 20320	++++++++####++++++++++	$2F78 12152	$4F78 20344	++++++++++++++++++++++++++++++++
244	$3360 13152	$5360 21344	++++++++####++++++++++	$3378 13176	$5378 21368	++++++++++++++++++++++++++++++++
245	$3760 14176	$5760 22368	++++++++####++++++++++	$3778 14200	$5778 22392	++++++++++++++++++++++++++++++++
246	$3B60 15200	$5B60 23392	++++++++####++++++++++	$3B78 15224	$5B78 23416	++++++++++++++++++++++++++++++++
247	$3F60 16224	$5F60 24416	++++++++####++++++++++	$3F78 16248	$5F78 24440	++++++++++++++++++++++++++++++++
248	$23E0 9184	$43E0 17376	++++++++####++++++++++	$23F8 9208	$43F8 17400	++++++++++++++++++++++++++++++++
249	$27E0 10208	$47E0 18400	++++++++####++++++++++	$27F8 10232	$47F8 18424	++++++++++++++++++++++++++++++++
250	$2BE0 11232	$4BE0 19424	++++++++####++++++++++	$2BF8 11256	$4BF8 19448	++++++++++++++++++++++++++++++++
251	$2FE0 12256	$4FE0 20448	++++++++####++++++++++	$2FF8 12280	$4FF8 20472	++++++++++++++++++++++++++++++++
252	$33E0 13280	$53E0 21472	++++++++####++++++++++	$33F8 13304	$53F8 21496	++++++++++++++++++++++++++++++++
253	$37E0 14304	$57E0 22496	++++++++####++++++++++	$37F8 14328	$57F8 22520	++++++++++++++++++++++++++++++++
254	$3BE0 15328	$5BE0 23520	++++++++####++++++++++	$3BF8 15352	$5BF8 23544	++++++++++++++++++++++++++++++++
255	$3FE0 16352	$5FE0 24544	++++++++####++++++++++	$3FF8 16376	$5FF8 24568	++++++++++++++++++++++++++++++++
256	$2BE0 11232	$4BE0 19424	++++++++####++++++++++	$2BF8 11256	$4BF8 19448	++++++++++++++++++++++++++++++++
257	$2FE0 12256	$4FE0 20448	++++++++####++++++++++	$2FF8 12280	$4FF8 20472	++++++++++++++++++++++++++++++++
258	$33E0 13280	$53E0 21472	++++++++####++++++++++	$33F8 13304	$53F8 21496	++++++++++++++++++++++++++++++++
259	$37E0 14304	$57E0 22496	++++++++####++++++++++	$37F8 14328	$57F8 22520	++++++++++++++++++++++++++++++++
260	$3BE0 15328	$5BE0 23520	++++++++####++++++++++	$3BF8 15352	$5BF8 23544	++++++++++++++++++++++++++++++++
261	$3FE0 16352	$5FE0 24544	++++++++####++++++++++	$3FF8 16376	$5FF8 24568	++++++++++++++++++++++++++++++++

Figure 5.9 HIRES Scanning Map Including Undisplayed Areas (4 of 4).

Table 5.2 Screen Memory Scanning Summary.

LOCATION	HBL	HBL'
SCREEN TOP	Last 16 of THIRD 40 and UNUSED 8	FIRST 40
SCREEN MIDDLE	Last 24 of FIRST 40	SECOND 40
SCREEN BOTTOM	Last 24 of SECOND 40	THIRD 40
VBL	Last 24 of THIRD 40	UNUSED 8 and first 32 of FIRST 40

TIME term switches low one video scanner clock (RAS' rising during PHASE 1) after V4 • V2 becomes true (see Figures 8.5 and 8.17). HIRES TIME switches high one video scanner clock after V4 • V2 becomes false. V4 • V2 identifies the last four lines of the TEXT display, and the one scanner clock delay makes the video data from RAM switch between HIRES and TEXT at the same time as other addressing inputs to the video ROM (see **MIXED MODE SWITCHING** in Chapter 8).

V4 • V2 actually identifies scan lines 160 through 191 and 224 through 261. The scanned memory switches to HIRES during the first part of VBL, back to TEXT for the second part of VBL, then back to HIRES for the top of the screen. The switching during VBL is, of course, not visible on the screen. This information is only important to those special applications where it is important to know what is scanned during the blanking periods.

The following is a reference list for scanned memory in the HIRES MIXED mode:

Line 0, HPE' + 1 thru
Line 160, HPE' — HIRES, Figure 5.9
Line 160, HPE' + 1 thru
Line 192, HPE' — TEXT, Figure 5.6
Line 192, HPE' + 1 thru
Line 224, HPE' — HIRES, Figure 5.9
Line 224, HPE' + 1 thru
Line 0, HPE' — TEXT, Figure 5.6

HPE' occurs during the first video scanner state of HBL. On the screen, the address switching point for HIRES MIXED mode comes just at the end of the display on the right side.

REFRESHING RAM IN THE APPLE IIe

The refresh requirement of 64K dynamic RAM is that every ROW address be accessed at least once every two milliseconds. To achieve refresh in the process of scanning RAM for video output, the address inputs to RAM had to be assigned very carefully. Only the **ROW address** assignments are pertinent to this discussion, and these are shown in Table 5.3.

Generally, the low order outputs of the video scanner are assigned to the RAM ROW address inputs. This is natural since the low order bits change at a high frequency. H0, H1, H2, SUM-A3, SUM-A4, and SUM-A5 are the six scanning address inputs which change at the highest frequency, and these are all RAM ROW address inputs. Every horizontal scan, these six terms will go through all of their possible states.

SUM-A6 is the next highest frequency RAM addressing input, but it is inappropriate as a ROW address input. This is because SUM-A6 and the lower order addressing terms form a 128-state word of which only 64 states are scanned during a horizontal period. The 64 scanned states remain constant during each of the four quarters of the complete vertical scan, so in each quarter, there are 64 unscanned states. For example, in the top third of the screen, the state of SUM-A6 low and all lower order bits high is never reached.

The final two RAM ROW address assignments are V0 and V1. These are not gated by the display mode, and they change at a high enough frequency to satisfy the refresh requirement. The gated VA, VB, and VC inputs are static during TEXT/LORES scanning, and thus inappropriate as RAM ROW address inputs.

The most significant refresh bit is V1, and V1 toggles back and forth every 32 horizontal scans. But VA, VB, and VC are not refresh bits, so for every state of V1 and V0, all the lower refresh bits go through their counts eight straight times (once for each state of VA-VB-VC). This means that the maximum time between refresh states is normally no more than 25 horizontal scans (32 − 7) or 1.59

Table 5.3 Video Scanner Row Address Assignments.

RAM Address	Scanner Input
RA0	H0
RA1	H1
RA2	H2
RA3	SUM-A3
RA4	SUM-A4
RA5	SUM-A5
RA6	V0
RA7	V1

milliseconds. At the end of VBL, however, there are six extra horizontal scans with V1 and V0 high. This means that there are sometimes 31 horizontal scans (25 + 6) or 1.97 milliseconds between refresh states. In all cases, the 2-millisecond maximum period between refresh states is satisfied.

As mentioned previously, some 64K dynamic RAM chips require only that A0—A6 of the RAM chips (RA7—RA1 of the Apple IIe RAM address bus) be refreshed. Since RA0—RA7 of the Apple IIe are completely refreshed every two milliseconds, RAM in the Apple can be either the 128- or the 256-cycle refresh type.

MEMORY MANAGEMENT

What device responds when the MPU accesses address $D000? The answer is all of the above. Depending on MMU soft switches and the peripheral slot INHIBIT' line, it might be the C1—DF ROM, motherboard high RAM bank 1, motherboard high RAM bank 2, auxiliary card high RAM bank 1, auxiliary card high RAM bank 2, or a device in any of the seven peripheral slots. Wow! Talk about your versatility . . . talk about your complexity.

When more than one device is capable of responding to a single address range, the range and device are said to be **bank switched**. Bank switching is necessary when the address ranges of all the devices that need to be addressed exceed the addressing range of the MPU, and when you hear of a 128K RAM, 16K ROM computer with a 64K MPU, you can guess that there is going to be some bank switching involved. Yet the nature and extent of the bank switching in the Apple makes no sense unless viewed in historical terms.

The original Apple II was designed at a time when 16K RAM chips and 2K ROM chips were state-of-the-art and 48K seemed like a lot of RAM. The 64K addressing range of the 6502 was neatly divided up into 48K of RAM, 4K of I/O, and 12K of ROM. In terms of memory, this computer evolved into the 64K RAM Apple II with 16K of RAM in Slot 0 and an 80-column display card in Slot 3, then into the operationally similar Apple IIe with its added capabilities of bank switching 64K of auxiliary card RAM. Apple IIe owners have thus inherited a complex system of memory management, and it can be a little puzzling, particularly for those new owners who weren't around for the evolution.

MMU Soft Switches

The bank switching and overall control of memory in the Apple IIe is a function of the MMU. The MMU contains 13 programmable **soft switches** that are set up by the controlling 6502 program. The state of these soft switches determines the overall configuation of Apple memory—which devices respond to which address ranges. Six of the MMU soft switches are concerned with bank switching between motherboard RAM and auxiliary card RAM. Four switches are concerned with determining whether high RAM or motherboard ROM responds to $D000—$FFFF addressing, and which bank of high RAM will respond to $D000—$DFFF. The remaining three switches determine whether motherboard ROM or peripheral cards respond to $C100—$CFFF addressing.

The soft switches are turned on and off by address commands decoded from the address bus. Some of the switches simply have an off address and an on address, but control of INTC8ROM and the high RAM configuration switches is more complex. In addition to address bus control, all MMU soft switches are reset (turned off) when a system reset is detected. Table 5.4 summarizes the address bus control of the MMU soft switches and gives the control situations resulting from system resets.

Some readers will be suprised by the Table 5.4 entries that show you can read the BANK1 and HRAMRD soft switches at $C011 and $C012. Apple doesn't document this capability, but I discovered it while investigating the operation of my Revision B Apple IIe (MMU marking 8310, 344-0010-B). Another thing Apple does not document is the existence of the INTC8ROM soft switch. The peripheral slot I/O STROBE' protocol is implemented for $C3XX and $C800—$CFFF motherboard ROM via this soft switch.

Configuring High Memory ($D000—$FFFF)

Addresses $D000—$FFFF in the Apple IIe are assigned primarily to ROM, but this 12K range can also be assigned to 16K of RAM. The capability of

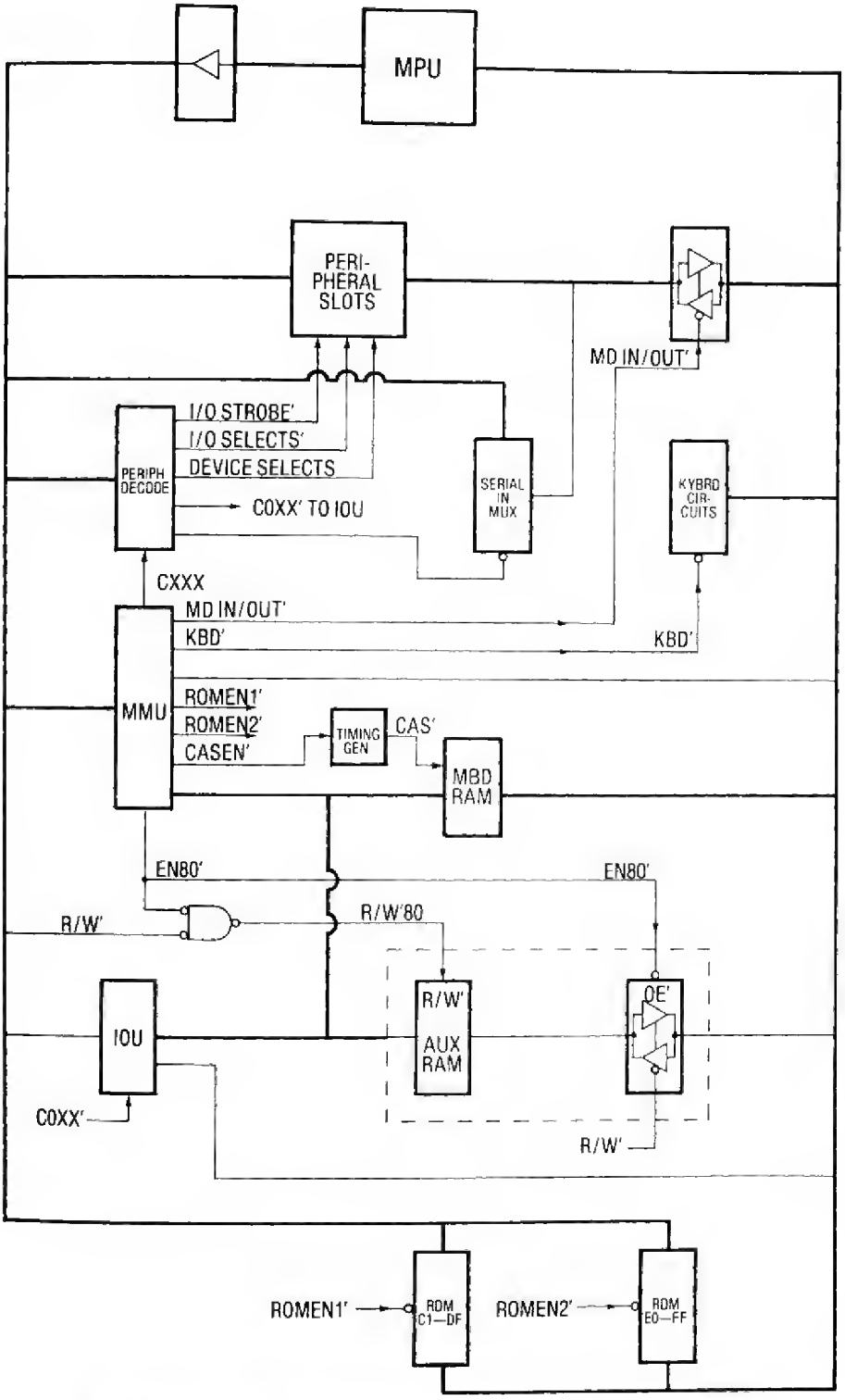


Figure 5.10 Distribution of MMU Data Bus Management Signals.

Table 5.4 Address Bus Control of MMU Soft Switches.

SOFT SWITCH	OFF ADDRESS	ON ADDRESS	READ ADDRESS	CONDITION AFTER RESET' (OFF)
80STORE	W$C000	W$C001	R$C018*	PAGE2 does not bank switch RAM
RAMRD	W$C002	W$C003	R$C013	Read from motherboard RAM
RAMWRT	W$C004	W$C005	R$C014	Write to motherboard RAM
INTCXROM	W$C006	W$C007	R$C015	Slot response to $C100—$CFFF
ALTZP	W$C008	W$C009	R$C016	Motherboard RAM read/write
SLOTC3ROM	W$C00A	W$C00B	R$C017	Motherboard ROM response to $C3XX
PAGE2	$C054	$C055	R$C01C*	Motherboard RAM read/write
HIRES	$C056	$C057	R$C01D*	PAGE2 does not switch $2000—$3FFF
BANK1**	A3'	A3	R$C011***	High RAM bank 2 response to $DXXX
HRAMRD**	(A1+A2) • (A1 • A2)'	A1 • A2+A1' • A2'	R$C012	$D000—$FFFF read from ROM
PRE-WRITE**	A0'+(R/W')'	A0 • R/W'	None	Reset
HRAMWRT' **	PRE-WRITE • R/W' • A0	A0'	None	$D000—$FFFF write to high RAM
INTC8ROM	$C3XX • SLOTC3ROM'	$CFFF	None	Slot response to $C800—$CFFF

NOTES:

R Preceeding address indicates read access only.
W Preceeding address indicates write access only.
* 80STORE, PAGE2, and HIRES are mechanized identically in the MMU and IOU. The MMU passes the state of 80STORE to MD7 when $C018 is read, and the IOU passes the state of PAGE2 or HIRES to MD7 when $C01C or $C01D is read.
** High RAM control addresses are in the $C08X range.
*** R$C011 reads the inversion of the BANK1 soft switch.

bank switching the high memory range between RAM and ROM gives the Apple IIe a bit of a split personality. On one hand, it is like the older cassette based computers with BASIC and the monitor instantly available in ROM. On the other hand, it is like the disk based computers which load and run any available operating system.

The RAM that is switched into the $D000—$FFFF range is referred to in this book as **high RAM**. It is designed to be operationally compatible with a 16K RAM card operating in Slot 0 of an Apple II. The concept and programming rules for high RAM came directly from the RAM card, so it is perhaps best to visualize high RAM as a peripheral which steals $D000—$FFFF from ROM. If high RAM is enabled, it responds to $D000—$FFFF; if high RAM is disabled, motherboard ROM responds to $D000—$FFFF.

Switching 16K of RAM into a 12K range creates an excess of 4K of RAM. For this reason, the 4K $DXXX range is bank switched between two 4K areas of RAM (see Figure 5.11). $E000—$FFFF addressing always causes access to the same 8K RAM locations when high RAM is enabled, but addressing $DXXX will cause access to high RAM bank 1 or bank 2 depending on the BANK1 soft

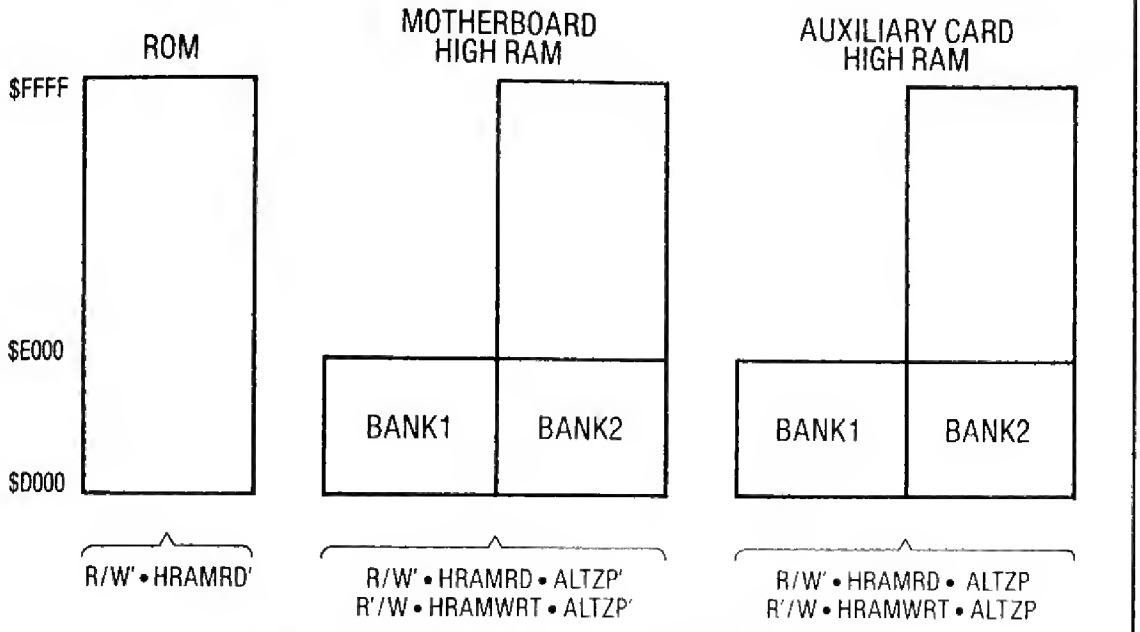


Figure 5.11 Selecting High Memory Configurations.

switch in the MMU. Bank 2 is the primary bank. It is selected by system resets, and programs generally utilize bank 2 rather than bank 1.

High RAM enabling and bank switching are controlled by four soft switches in the MMU. Program control of these soft switches is identical to that of four configuration flip-flops on the 16K RAM card.* All control accesses are in the $C08X range which is the Slot 0 DEVICE SELECT' range of an Apple II. Address bus manipulation of the high RAM soft switches is as follows:

1. A3 controls the 4K bank selection. $C080—$C087 resets the **BANK1** soft switch, enabling bank 2. $C088—$C08F sets the BANK1 soft switch, enabling bank 1.
2. A0 and A1 control the **HRAMRD** soft switch. Access to $C080, $C083, $C084, $C087, $C088, $C08B, $C08C, or $C08F sets HRAMRD, enabling reading from high RAM. Access to $C081, $C082, $C085, $C086, $C089, $C08A, $C08D, or $C08E resets HRAMRD, disabling reading from high RAM.
3. Writing to high RAM is enabled when the **HRAMWRT'** soft switch is reset. The controlling MPU program must set the **PRE-WRITE** soft switch before it can reset HRAMWRT'. **PRE-WRITE** is set by odd read access in the $C08X range. It is reset by even read access or any write access in the $C08X range. HRAMWRT' is reset by odd read access in the $C08X range when PRE-WRITE is set. It is set by even access in the $C08X range. Any other type of access causes HRAMWRT' to hold its current state.
4. When a system reset occurs, all MMU soft switches are reset (turned off). High RAM is disabled for reading and enabled for writing. PRE-WRITE is reset, and bank 2 is selected. Since a reset occurs when the Apple is turned on, the Apple IIe always powers up with high RAM disabled for reading.
5. When high RAM is enabled, $D000—$FFFF addressing causes access to either motherboard high RAM or auxiliary card high RAM as controlled by the ALTZP soft switch. Switching between motherboard RAM and auxiliary card RAM is discussed in the next section.

PRE-WRITE and WRITE can be thought of as a write counter which counts odd read accesses in the $C08X range. The counter is set to zero by even or write access in the $C08X range. If the write counter reaches the count of 2, writing to high RAM becomes enabled. From that point, writing will stay enabled

*For a discussion of the 16K RAM card, refer to Chapter 5 of *Understanding the Apple II* by Jim Sather, Quality Software, 1983.

until an even access is made in the $C08X range. This means there is a feature of RAM card control not documented by Apple: write access to an odd address in the $C08X range controls HRAMRD without affecting the state of HRAMWRT'.

The high RAM control characteristics are summarized in Table 5.5. There are two address commands possible for every function. The programming convention is to use addresses $C080—$C083 and $C088—$C08B.

The only address range at which RAM is never accessed in the Apple IIe is the 4K range, $CXXX. When bank 1 of high RAM is being accessed, the MMU converts the $DXXX address on the address bus to a $CXXX address on the multiplexed RAM address bus. $DXXX is changed to $CXXX by changing A12 from high to low, so the A12 input to RA4/COLUMN is forced low in the MMU when BANK1 is set and an address in the $DXXX range is on the address bus. The equation for the COLUMN input to RA4 is therefore

$$A12 \bullet (BANK1 \bullet DXXX)'.$$

Though program control of high RAM in the Apple IIe is identical to that of the Slot 0 16K RAM card in the Apple II, there are some operational differences. First, the 16K RAM card does not respond to system resets. It is set to read disable, write enable bank 2 at power up, and from that point can only be reconfigured by program control. High RAM in the Apple IIe is set to read disable, write enable bank 2 by any system reset, not just the one that occurs at power up.

A second difference is that the 16K RAM card is not automatically disabled when another peripheral card pulls the INHIBIT' line low. A particularly nettlesome associated fact is that the 16K RAM card will not release the $F800—$FFFF address range under any circumstances, even when its RAM is disabled for reading and writing.* When RAM is disabled, a ROM on the RAM card hogs this critical range. When INHIBIT' is low in the Apple IIe, all motherboard RAM, auxiliary card RAM, and motherboard ROM is disabled, including high RAM.

The data bus management signals with which the MMU configures high memory are **CASEN'**, **EN80'**, **ROMEN1'**, and **ROMEN2'**. All of these signals are gated by INHIBIT', so no matter what the configuration of high memory, any peripheral card can steal $D000—$FFFF by pulling INHIBIT' low.

Switching between Motherboard and Auxiliary Card RAM

Switching between memory banks presents a problem if the program doing the switching resides in one of the memory banks being switched. The problem is that when the switching command is executed, the MPU starts fetching the program from the newly active memory bank, but the program that is supposed to be executed resides in the

*This is only true of the 16K RAM card manufactured by Apple Computer, Inc., and of RAM cards which are very close imitations of Apple's card. Some alternate source 16K RAM cards will release the $F800—$FFFF range when RAM on the card is disabled.

Table 5.5 High RAM Address Bus Commands.

BANK 2	BANK 1	ACTION	
$C080 $C084	**$C088** $C08C	WRTCOUNT = 0*, WRITE DISABLE	READ ENABLE
R$C081 R$C085	**R$C089** R$C08D	WRTCOUNT = WRTCOUNT + 1*	READ DISABLE
W$C081 W$C085	**W$C089** W$C08D	WRTCOUNT = 0*	READ DISABLE
$C082 $C086	**$C08A** $C08E	WRTCOUNT = 0*, WRITE DISABLE	READ DISABLE
R$C083 R$C087	**R$C08B** R$C08F	WRTCOUNT = WRTCOUNT + 1*	READ ENABLE
W$C083 W$C087	**W$C08B** W$C08F	WRTCOUNT = 0*	READ ENABLE

*Writing to high RAM is enabled when WRTCOUNT reaches 2.

previously active memory bank. One way to get around this is to prearrange the contents of the newly active bank so that there is a program at the switch point which is designed to accept flow from the previously active bank. This is not an insurmountable task for the computer programmer, but it is a nuisance. A better solution, when possible, is to design the application so that the switching program resides outside of the memory being switched.

Apple went to some lengths to ease the programmer's task in bank switching between motherboard RAM and auxiliary card RAM in the Apple IIe. For one thing, they provided a transfer routine in ROM which transfers program control between motherboard resident and auxiliary card resident routines. Therefore, any time high RAM is disabled for reading, routines residing outside of the memory being switched are available for switching between motherboard RAM and auxiliary card RAM. Use of the firmware transfer routine is described on pages 29—31 of the *Apple II Extended 80-Column Text Card Supplement for IIe Only* and on pages 76—79 of the *Apple II Reference Manual for IIe Only.*

Another thing that Apple did to ease the programmer's bank switching problems was to divide RAM into several separately switched groups so that programs residing in one area of RAM could switch other areas of RAM. Instead of one soft switch which switches all of RAM access between the motherboard and the auxiliary card, there are six MMU soft switches which switch various operational areas of RAM. The sense of the switches is generally such that when a switch is set (on), auxiliary card RAM is enabled, and when the switch is reset (off), motherboard RAM is enabled. The functions of these six switches are graphically depicted in Figure 5.12 and briefly described here. Refer to Table 5.4 for the control addresses of the soft switches.

RAMRD and **RAMWRT** switch the $200—$BFFF range between motherboard RAM and auxiliary card RAM. RAMRD set enables auxiliary card memory for reading, and RAMWRT set enables auxiliary card memory for writing. If 80STORE is set, RAMRD and RAMWRT do not affect $400—$7FF, and if 80STORE and HIRES are both set, RAMRD and RAMWRT do not affect $400—$7FF or $2000—$3FFF.

80STORE, **PAGE2**, and **HIRES** bank switch the primary display pages, $400—$7FF and $2000—$3FFF, between motherboard RAM and auxiliary card RAM. If 80STORE is set and HIRES is reset, then PAGE2 switches between motherboard RAM and auxiliary card RAM for reading and writing in the $400—$7FF range. If 80STORE is set and HIRES is set, then PAGE2 switches beween motherboard RAM and auxiliary card RAM for reading and writing in the $400—$7FF and $2000—$3FFF ranges. PAGE2 set selects auxiliary card RAM, and PAGE2 reset selects motherboard RAM. If 80STORE is reset, then RAMRD and RAMWRT will bank switch the $400—$7FF and $2000—$3FFF ranges along with the rest of the $200—$BFFF range.

80STORE, PAGE2, and HIRES are mechanized identically in the MMU and the IOU. They are used in the MMU for bank switching display memory as described here. They are used in the IOU to control the display mode of the Apple IIe. As noted previously, when 80STORE is set, the display mode control functions of PAGE2 are disabled.

ALTZP switches the $0—$1FF range and, if high RAM is enabled, the $D000—$FFFF range between motherboard RAM and auxiliary card RAM. ALTZP set selects auxiliary card RAM, and ALTZP reset selects motherboard RAM.

The overall scheme represented by these soft switch capabilities is logical. High RAM is switched separately from the main body of RAM because these two RAM groups are addressed at separate non-overlapping address ranges. Memory pages 0 and 1 of RAM receive special treatment because of their special 6502 functions. Display scanned memory receives special treatment to ease the task of programming DOUBLE-RES displays. Some programming scenarios should illustrate the pros and cons of the way RAM switching is divided.

scenario #1: A machine language program resides in high RAM, motherboard ROM, or peripheral card high memory, and uses both motherboard and auxiliary card low RAM for data storage.

The program can easily switch between motherboard and auxiliary card RAM via RAMRD and RAMWRT. The program's critical zero page pointers and subroutine return link information on the stack are not switched because RAMRD and RAMWRT do not switch $0—$1FF. Pages 0 and 1 of RAM usually hold information critical to the program being executed, even if the program resides in ROM. All programs that bank switch RAM need to take special cognizance of these two memory pages, and the Apple IIe design allows the programmer to do so because $0—$1FF switching is separate from the main body of RAM in the Apple.

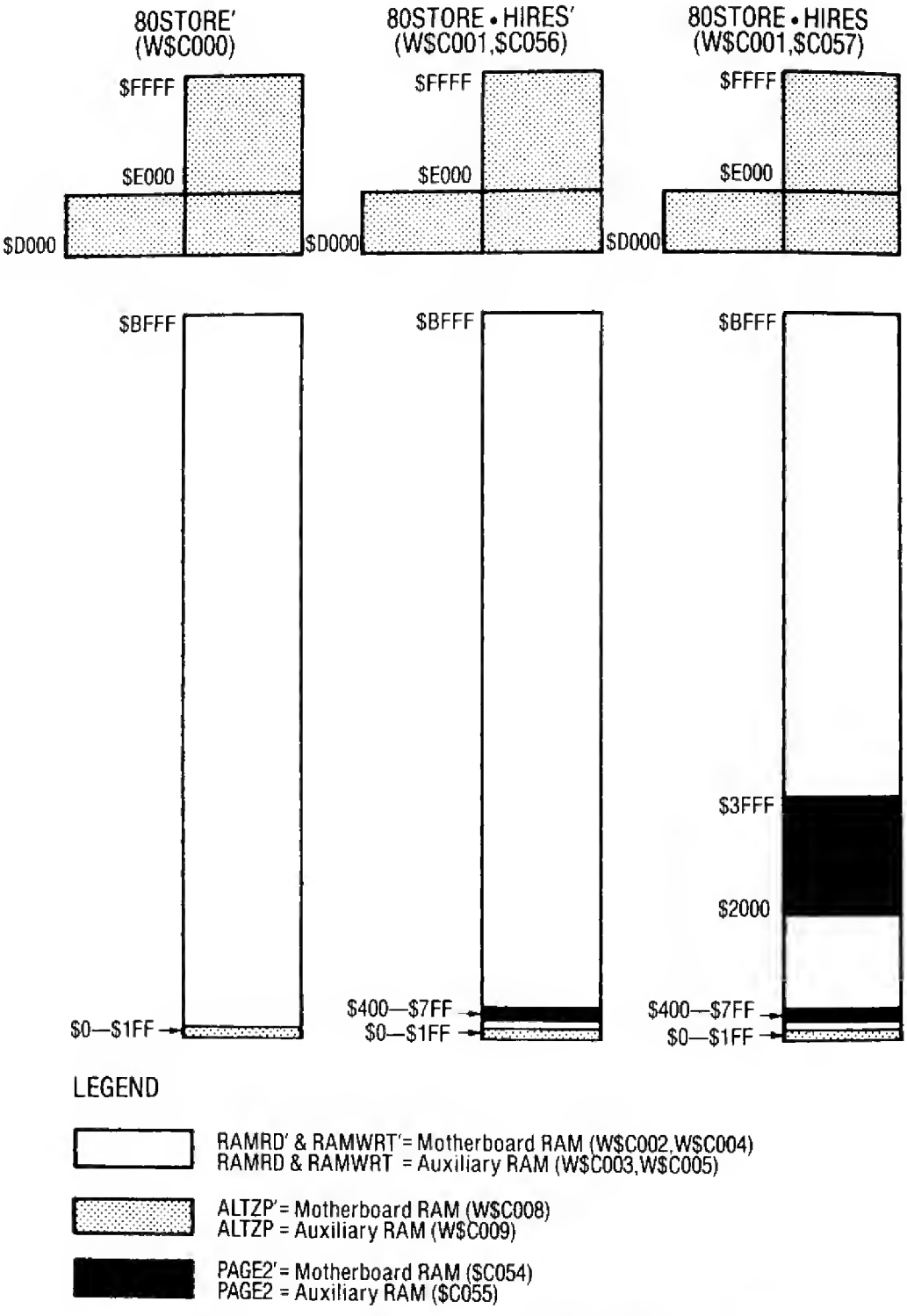


Figure 5.12 Motherboard/Auxiliary Card RAM Bank Switched Address Ranges.

Example:

```
ORG $D000     HIGH RAM RESIDENT PROGRAM
STA $C003     SET RAMRD
STA $C004     RESET RAMWRT
LDA $8000     FETCH DATA FROM AUX $8000
STA $8000     STORE DATA AT MRBRD·$8000
```

scenario #2: A machine language program resides in lower motherboard RAM, and uses motherboard and auxiliary card high RAM for data storage. The program is initialized and normally runs with motherboard memory pages 0 and 1 enabled.

This situation is not particularly well supported by the Apple IIe design. The program can enable auxiliary card high RAM via ALTZP, HRAMRD, and HRAMWRT', but when auxiliary card high RAM is enabled, the program loses access to its Page 0 pointers and subroutine return link information. The program must therefore make its access to auxiliary card high RAM, then reset ALTZP before accessing pages 0 and 1. Alternately, it can transfer critical page 0 and 1 information from motherboard RAM to auxiliary card RAM before setting ALTZP, and it can transfer critical page 0 and 1 information from auxiliary card RAM to motherboard RAM before resetting ALTZP.

Example:

```
ORG $1000     LOW RAM RESIDENT PROGRAM
STA $C009     SET ALTZP
LDX $BA       WHOOPS! I NEEDED A MOTHER-
              BOARD POINTER
STA ($44),Y   OUCH! MY INDIRECTION IS
              GUMMED UP
RTS           I WANT MY MOTHER
```

scenario #3: A machine language program resides in low RAM and maintains an 80-column text display.

The text map is stored alternately in auxiliary card RAM and motherboard RAM at addresses $400—$7F7, so the program needs to make regular access to both the auxiliary card and the motherboard without disrupting program flow. This is done simply by setting 80STORE, resetting HIRES, and using PAGE2 to switch MPU access back and forth between auxiliary card RAM and motherboard RAM. Program flow outside of the $400—$7FF range is not affected in any way. The identical method is used for maintaining an 80-block LORES display. A 560-point HIRES display can be similarly maintained by setting 80STORE and HIRES and using PAGE2 to switch $2000—$3FFF access between auxiliary card RAM and motherboard RAM.

The situation becomes more difficult if the program needs to maintain a primary and a secondary DOUBLE-RES display, and switch between them via PAGE2 with 80STORE reset. The problem here is that MPU access to the secondary display areas ($800—$BFF and $4000—$3FFF) is switchable only via RAMRD and RAMWRT. The task can be accomplished by storing the controlling program redundantly in motherboard and auxiliary card RAM, but things will be a lot easier if this sort of program resides in high RAM.

Example:

```
ORG $1000     LOW RAM RESIDENT PROGRAM
STA $C001     SET 80STORE
STA $C055     PAGE2 = AUX = EVEN
              TEXT POSITION
STA (BASL),Y  STORE ASCII TO TEXT MAP
```

scenario #4: A programmer wants to utilize auxiliary card memory for data storage in his Applesoft or Integer BASIC application.

The programmer is, as they say in the Navy, S-O-L (Surely Out of Luck). BASIC variable structure does not support the fixed address access to memory required by bank switching. Additionally, the interpreter in high memory, the program in low RAM, the variables in low RAM, and the page 0 and 1 values tend to get lost when you enable the auxiliary card RAM. BASIC programs can manipulate 80STORE, HIRES, and PAGE2 to access $400—$7FF and $2000—$3FFF of auxiliary card RAM via PEEK, POKE, HPLOT, PLOT, HLIN, VLINE, and PRINT commands. Outside of the 80STORE areas, it is most likely that any BASIC access to auxiliary memory would be through CALLs to machine language routines.

Example:

```
10 POKE -16383,0 : REM SET 80STORE
20 POKE -16299,0 : REM PAGE2 = AUX =
   EVEN
30 POKE 1024,193 : REM DISPLAY "A" AT
   TOP LEFT
```

The data bus management signals with which the MMU switches between motherboard and auxiliary card RAM are **CASEN'** and **EN80'**. CASEN' is low when the MPU is accessing motherboard RAM, and EN80' is low when the MPU is accessing auxiliary card RAM. Both CASEN' and EN80' are gated by INHIBIT' so any peripheral card can steal RAM addressing from motherboard or auxiliary card RAM by pulling INHIBIT' low.

Configuring the I/O Range ($C000—$CFFF)

The 4K $C000—$CFFF range in the Apple IIe is assigned to I/O. Motherboard ROM, however, can steal $C100—$CFFF from I/O and it does so on a regular basis. Program control of response to $C100—$CFFF is via three MMU soft switches, **INTCXROM***, **SLOTC3ROM**, and **INTC8ROM**.

The $C100—$CFFF range of I/O includes the peripheral slot I/O SELECT' and I/O STROBE' signals. These signals are generally used to enable peripheral slot ROM, and INTCXROM, SLOTC3ROM, and INTC8ROM are thought of as switching between slot ROM and internal (motherboard) ROM. These soft switches have no effect on the $C0XX range. **There is no hardware or software means of stealing $C0XX from I/O in the Apple IIe.** A corollary to this fact is that the DEVICE SELECTs of the peripheral slots cannot be inhibited in the Apple IIe. In particular, peripheral cards that respond to DEVICE SELECT', but not I/O SELECT' or I/O STROBE', can be operated in Slot 3, even when a RAM/80-column card is installed in the auxiliary slot.

INTCXROM switches the $C100—$CFFF range between internal and slot ROM, and SLOTC3ROM switches the $C3XX range. Both INTCXROM and SLOTC3ROM affect $C3XX (the Slot 3 I/O SELECT' range), and if either soft switch is set to internal, the motherboard C1—DF ROM will respond to $C3XX access. The following truth table shows the control status resulting from the four possible combined states of INTCXROM and SLOTC3ROM.

INTCXROM	SLOTC3ROM	$C100-$C2FF $C400-$CFFF	$C300-$C3FF
reset	reset	slot	internal
reset	set	slot	slot
set	reset	internal	internal
set	set	internal	internal

You will find no reference to the INTC8ROM soft switch in Apple literature. You will find, however, that it is possible for motherboard ROM to steal response to $C800—$CFFF without setting INTCXROM. The $C800—$CFFF range is assigned to motherboard ROM (INTernal) anytime SLOTC3ROM is reset and an access is made to $C3XX. From that point, $C800—$CFFF will stay assigned to motherboard ROM until an access is made to $CFFF or until the MMU detects a system reset. The circuit mechanization which fits this functional reality is that of an unreadable soft switch, set by access to $C3XX with SLOTC3ROM reset, and reset by access to $CFFF or an MMU reset.

As it is with the $C3XX range, joint control of the $C800—$CFFF range is an OR function if INTernal is considered true and SLOT is considered false. By this it is meant that if INTCXROM OR SLOTC3ROM is configured for internal response, then access to $C3XX results in ROMEN1' low (active) and Slot 3 I/O SELECT' high (inactive). In the same vein, if INTCXROM OR INTC8ROM is configured for internal response, then access to $C800—$CFFF results in ROMEN1' low (active) and I/O STROBE' high (inactive).

The INTC8ROM mechanization follows protocol for a Slot 3 peripheral card that responds to I/O SELECT' and I/O STROBE'. This is consistent with the Apple IIe philosophy of emulating an Apple II with 80-column card installed in Slot 3. More information about the I/O STROBE' protocol is given in Chapter 7.

INTCXROM, SLOTC3ROM, and INTC8ROM allow MPU access to the lower half of the C1—DF ROM. INTCXROM is used to gain access to a number of routines including the extended monitor routines and built-in diagnostic routines. SLOTC3ROM and INTC8ROM are used to gain access to the 80-column firmware.

Like all MMU soft switches, INTCXROM, SLOTC3ROM, and INTC8ROM are reset when a system reset is detected. This enables all I/O decoding except for Slot 3 I/O SELECT' ($C3XX). After the hardware reset, the motherboard firmware reset handler sets SLOTC3ROM if no RAM card is installed in the auxiliary slot, and resets SLOTC3ROM if a RAM card is installed in the auxiliary slot.

The data bus management signals with which the MMU switches between internal or slot response to $C100—$CFFF are **CXXX** and **ROMEN1'**. ROMEN1' is gated by INHIBIT', so peripheral cards can steal $C100—$CFFF by disabling slot response and bringing INHIBIT' low.

KBD' and MD IN/OUT'

There are two MMU data bus management signals which have nothing to do with RAM. These are **KBD'**, the keyboard enable signal, and **MD IN/OUT'** the peripheral slot bus driver direction control signal.

*Apple refers to INTCXROM as SLOTCXROM, but slot ROM is disabled when the "CX" switch is set. The "CX" switch can be correctly referred to as the SLOTCXROM' switch, but I feel that INTCXROM is more descriptive. There appears to have been some miscommunication at Apple about the operation of this switch. Operational descriptions on pages 133 and 214 of the *Apple II Reference Manual for IIe Only* are demonstrably inaccurate.

The MMU pulls KBD' low during PHASE 0 of a read access to the $C000—$C01F range. This uninhibitable action causes the contents of the keyboard ROM to be placed on the data bus for reading by the MPU. Normal programming convention is to read the keyboard input at $C000, and the fact that you can read it at $C01X is definitely not documented by Apple. Nevertheless, with the MMU currently being manufactured, the KBD' signal responds to read access at $C000—$C01F.

MD IN/OUT' is normally low, and this causes the peripheral slot bidirectional driver to normally present a high impedance to the data bus. MD IN/OUT' goes high during PHASE 0 of

1. Any read to $C020—$C0FF (enables MPU or DMA device to read $C06X serial inputs, DEVICE SELECT' gated peripheral card inputs, or outlandish peripheral cards which respond with data to $C020—$C08F addressing).
2. Any read to $C100—$CFFF with slot response enabled by INTCXROM, SLOTC3ROM, and INTC8ROM (enables MPU or DMA device to read I/O SELECT' and I/O STROBE' gated peripheral card inputs).
3. Any read with INHIBIT' low (enables MPU or DMA device to read from peripheral card memory when motherboard and auxiliary card memory are inhibited).
4. Any write with DMA' low (enables DMA device to write data to devices on the data bus).

The MD IN/OUT' control may seem complex, but when you examine the requirements of the Apple IIe memory map, MD IN/OUT' is seen to correctly control the direction of data flow to and from the peripheral slots and serial input multiplexor. The one unusual point about MD IN/OUT' mechanization is that read access to $C000—$C01F with INHIBIT' low causes MD IN/OUT' to go high, even though the keyboard input is not deactivated by INHIBIT'. As a result, read access to $C000—$C01F with INHIBIT' low will cause the peripheral data bus driver to compete with (and overwhelm) the keyboard ROM and IOU for control of the data bus. This is not a great problem for peripheral card designers, but it does mean that inhibiting peripheral cards must bring INHIBIT' high during read access to $C000—$C01F if the MPU is to read the keyboard.

The MMU Functional Diagram

Figures 5.13a and 5.13b are a functional diagram of the MMU. Figure 5.13a shows address decoding, reset detection, MPU address multiplexing, and the MMU soft switches. Figure 5.13b shows the logical generation of data bus management signals from address decoded signals, the states of the MMU soft switches, and the R/W', DMA', and INHIBIT' inputs to the MMU. These drawings are meant only to show MMU functions, not correct implementation details. I drew these figures to clearly illustrate MMU operation and have no way of knowing such details as the exact arrangement of logic gates.

The MMU drawings summarize points about MMU operation that are made in *Understanding the Apple IIe*, and they serve as an MMU quick reference for people who like diagrams like this (I do). Some explanations and discussion points follow.

1. RESET' is not an input to the MMU because there is a shortage of available pins created by full connection to the address bus and the multiplexed RAM address bus. The MMU detects system resets from three Page 1 accesses followed by $FFFC on the address bus. The MMU will wrongly detect a system reset if a Page 1 resident program causes the MPU to access $FFFC, perform a "JMP ($FFFC)", or vector to $FFFC via an RTS or RTI instruction.
2. There appears to be a window during which the address bus is monitored for commands which determine soft switch states. If a soft switch command is held valid on the address bus for about 40 nanoseconds or more during the window, the affected soft switch will respond to the command. As well as I can determine, this window is PHASE0 • (Q3 + RAS'') which is all but the last 14M period of PHASE 0. The 6502 address valid period completely overlaps this window.
3. When the MPU reads the state of an MMU soft switch, the MD7 enable gate appears to be PHASE0 + PHASE0' • Q3 • RAS' which is PHASE 0 and the first 14M period of the following PHASE 1.
4. The four high-RAM configuration soft switches are programmed identically to the four configuration flip-flops on a 16K RAM card.
5. After an MMU reset, the MPU will read all soft switches except BANK1 as zero. Figure 5.13a shows a BANK1 soft switch whose inverted state is readable by the MPU. A functional equivalent would be a BANK2 soft switch which is set when an MMU reset occurs and whose non-inverted state is read by the MPU. The depiction chosen for Figure 5.13a is closer to the 16K RAM card hardware, and it follows the general MMU rule of resetting soft switches when a system reset occurs.

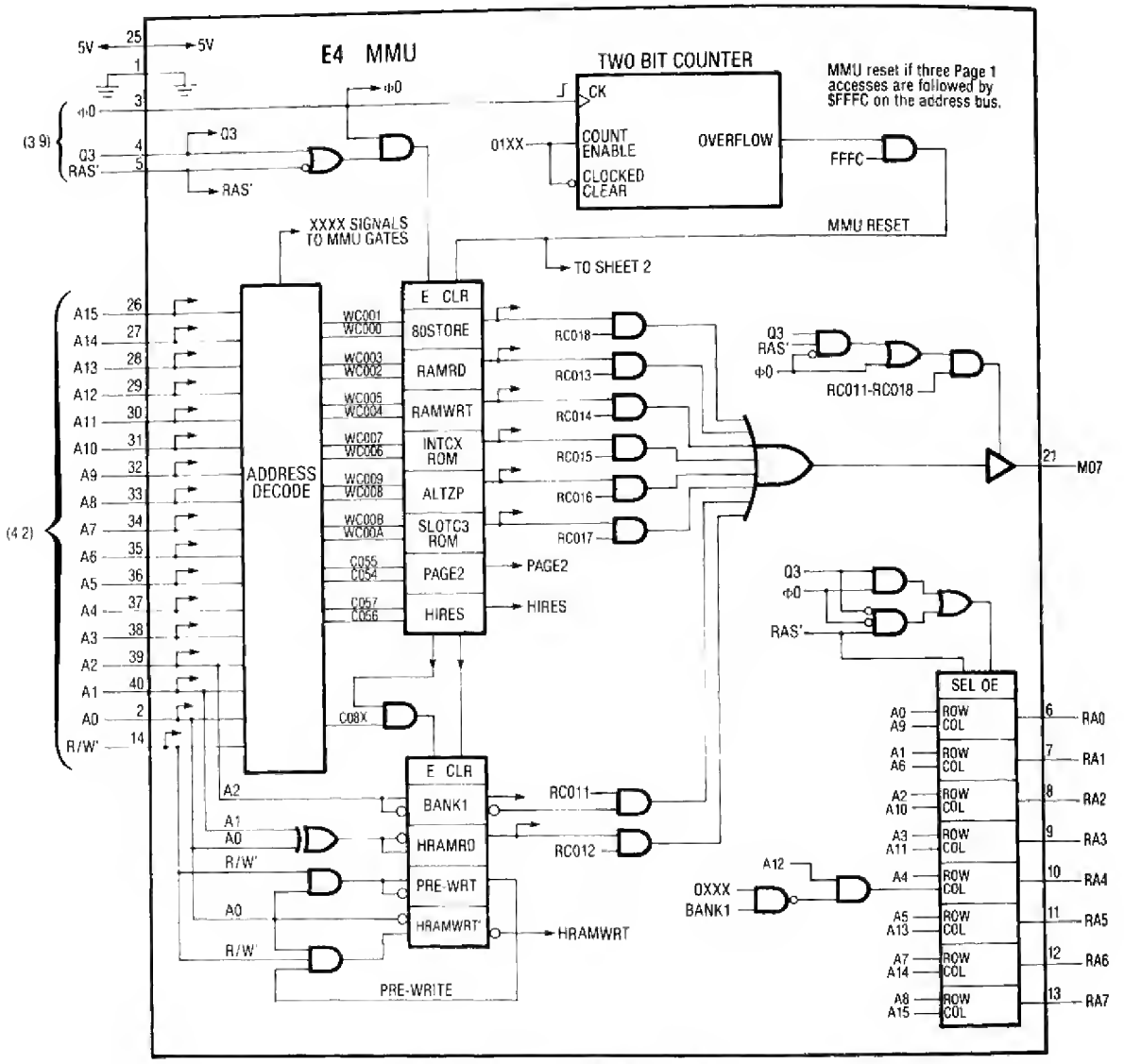
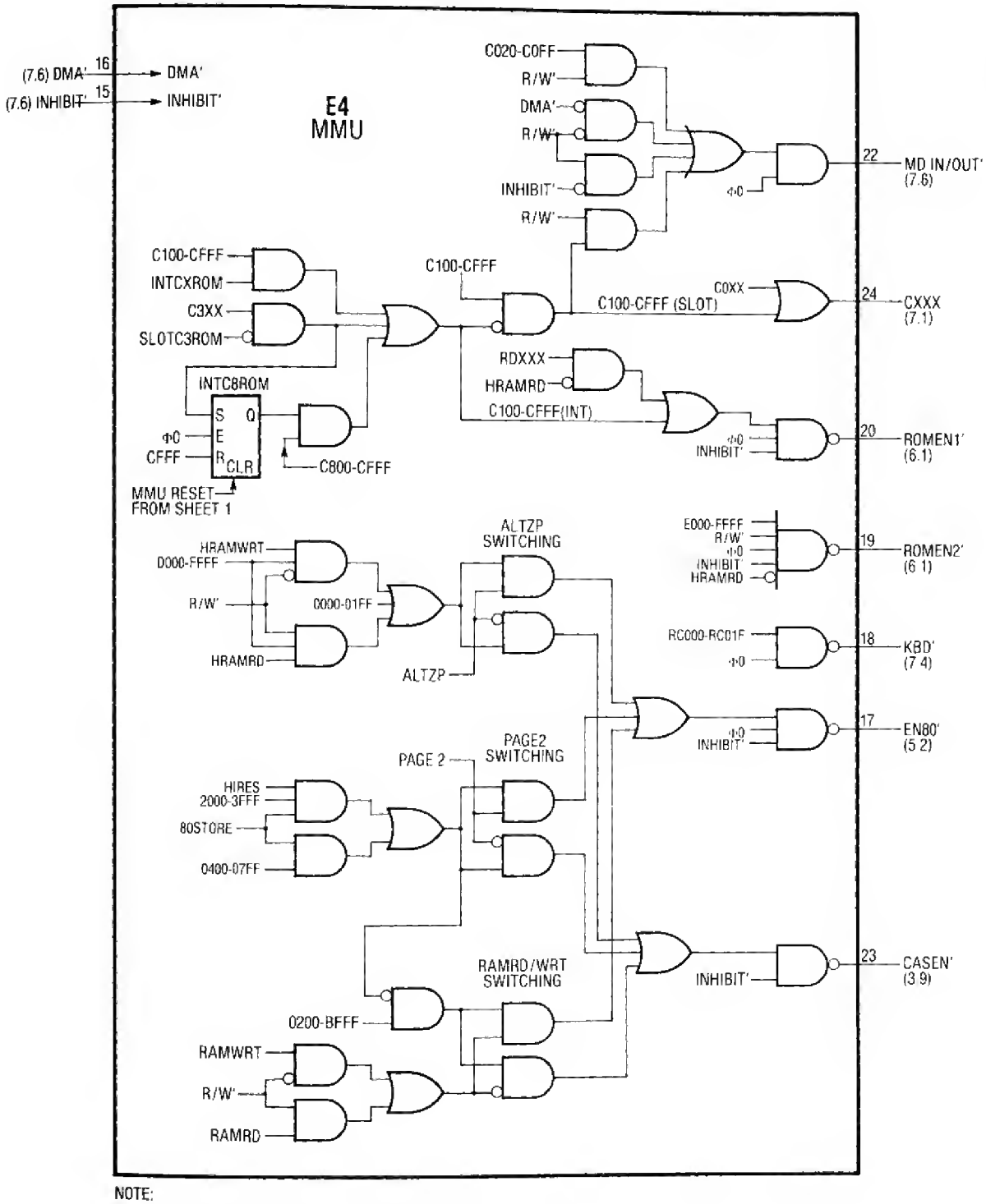


Figure 5.13a The MMU (1 of 2).

6. RAS' selects between ROW and COLUMN addressing at the MPU address multiplexor. The tri-state enable window for the MMU multiplexed RAM address output appears to be PHASE0' • Q3' • RAS' + PHASE0 • Q3 which is the last 14M period of PHASE 1 and the first four 14M periods of PHASE 0.
7. RA4 COLUMN addressing has special logic for switching between bank 1 and bank 2 of high RAM.
8. ROMEN1', ROMEN2', EN80', KBD', and MD IN/OUT' are all gated by PHASE 0. CASEN' is not gated by PHASE 0 in the MMU, but it is gated by PHASE 0 in the timing HAL.
9. All memory enabling signals—ROMEN1', ROMEN2', CASEN', and EN80'—are gated by INHIBIT' from the peripheral slots.
10. CASEN' and EN80' are mutually exclusive. When RAM is being accessed, either CASEN' or EN80', but not both, will be activated.
11. KBD' is gated by RC000—RC01F rather than just the expected RC00X.
12. MD IN/OUT' goes high during any INHIBIT' low read rather than the expected R0000—RBFFF and RC020—RFFFF.
13. Write access to $C100—$CFFF with the addressed location configured for motherboard ROM response causes ROMEN1' to fall. As a



NOTE:
[1] Address range signals decoded from address bus.

Figure 5.13b The MMU (2 of 2).

result, this programming action will cause the C1—DF ROM to compete with the MPU for control of the data bus. This is the only time ROMEN1′ or ROMEN2′ is activated by write access.

The data bus management signals are logically mechanized to perform the memory management functions described in this chapter. Figure 5.13b shows this logic for those readers comfortable with logic symbols. Tables 5.6 and 5.7 contain the same information for those readers more comfortable with cause and effect tables. Table 5.6 shows when the signals that gate data to the data bus become active, and Table 5.7 shows when MD IN/OUT′ becomes active.

MMU Signal Propagation Delay

The MMU and IOU are MOS ICs like the 6502, RAM, and ROM chips. These MOS ICs perform very complex and extensive logic functions, but compared to the bipolar ICs like TTL chips and the timing HAL, MOS ICs are slow. Whereas you might expect 5—25 nanosecond signal propagation delay in an LSTTL chip, you would likely expect 30—125 nanosecond propagation delays in MOS ICs. These approximations are admittedly vague, but you should get the idea that the speed of the Apple is limited by the MOS chips, not the TTL chips.

The MMU signal delay specifications have not been released by Apple Computer Inc. If we could see them, they would probably show minimum and maximum timing durations referenced to rising and falling edges of PHASE 0, Q3, RAS′, and the address bus. I have observed the MMU signal delays in my Apple IIe with an oscilloscope, and I found them generally to be 40—100 nanoseconds. As a rule of thumb, the MMU signal delays are comparable in width to one 14M period.

One MMU specification I think Apple should publish is the maximum delay of address information from the address bus input to the multiplexed address bus output. This specification determines the time by which DMA devices must set up their address if RAS′ falling is to properly strobe the ROW address to RAM and the IOU. This delay is about 82 nanoseconds in my Apple IIe.* My feeling is that DMA devices are probably safe if they set up the address bus at or before RAS′ rises during PHASE 1. It has, however, been indicated to me that Apple only tests the MMU to perform correctly with worst case performance of the 6502A. Therefore, the only way a DMA device can be guaranteed to operate in the Apple IIe is if its address is set up on the address bus no later than 210 nanoseconds after PHASE 0 falls (5 nsec LS02 delay + 65 nsec 6502 PHASE 2 delay + 140 nsec 6502 address setup). This is unrealistically conservative since an MMU would have to have a 349-nanosecond address bus to RAM address bus delay to perform this poorly.

*I measured this by grounding the READY and DMA′ lines and injecting a variable width pulse to A0 of the address bus from a pulse generator triggered by WNDW′.

RAM TIMING IN THE APPLE IIe

Most aspects of RAM timing have already been covered in various other related discussions. The intention here is to reinforce the timing details of basic RAM access.

Figure 5.14 shows some basic details of video scanner read and 6502 read/write access to motherboard RAM. The RAM access is controlled by the RAS′ and CAS′ timing signals, and these fix the RAM access rate at 2 MHz. Even though 200-nanosecond RAM can be accessed faster than this, no DMA device could access Apple IIe motherboard RAM any faster unless higher frequency substitutes for RAS′ and CAS′ were injected from an auxiliary card. RAS′ strobes the ROW address to the RAM chips, and CAS′ strobes the COLUMN address to the RAM chips and initiates the data transfer.

Most of the RAM timing is straightforward. Write data from the 6502 is set up well before CAS′ falls during PHASE 0. Read data becomes valid well before PHASE 0 rises to latch video data, and well before PHASE 2 falls to clock the transfer of read data to the 6502. One thing that is not straightforward is the RAM data hold time after CAS′ rises during MPU read access to RAM. It is highly possible that the RAM chip will not hold the data past the falling edge of PHASE 2, but the data transfer is accomplished anyway because of the slow bleed off of data from the floating data bus. The data bus and all of its extensions hold data for a long time when they float. Insuring the correct transfer of RAM read data to the 6502 is just one example of the long bleed off time determining Apple IIe operational characteristics.

In the write cycle, the video scanner makes its read access to RAM during PHASE 1 just as in the read cycle. Nothing interrupts the scanner access to RAM in an unmodified Apple. However, the video data on the data bus is not routed to the peripheral cards during PHASE 1 of write cycles.

Table 5.6 MMU Data Bus Communication Enable Signals.

ADDRESS RANGE	INPUTS R/W'	RAMRD	RAMWRT	HRAMRD	HRAMWRT'	80STORE	HIRES	PAGE2	ALTZP	INTCXROM	SLTC3ROM	INTC8ROM	PHASE	INHIB'	DATA BUS GATING SIGNALS CASEN'	RDMEN80'	ROMEN1'	ROMEN2'	CXROM'	KBD'
$0000-$01FF	.	.	.	.	.	.	.	.	Ø	.	.	.	.	H	L	H	H	H	L	H
	.	.	.	.	.	.	.	.	1	.	.	.	H	H	H	L	H	H	L	H
TABLE A	R	Ø	.	.	.	.	.	.	.	.	.	.	.	H	L	H	H	H	L	H
$0200-$03FF,	R	1	.	.	.	.	.	.	.	.	.	.	H	H	H	L	H	H	L	H
$0800-$1FFF,	W	.	Ø	.	.	.	.	.	.	.	.	.	.	H	L	H	H	H	L	H
$4000-$BFFF	W	.	1	.	.	.	.	.	.	.	.	.	H	H	H	L	H	H	L	H
$0400-$07FF	.	.	.	.	.	Ø	.	.	.	.	.	.	.	.	SEE TABLE A					
	.	.	.	.	.	1	.	Ø	.	.	.	.	.	H	L	H	H	H	L	H
	.	.	.	.	.	1	.	1	.	.	.	.	H	H	H	L	H	H	L	H
$2000-$3FFF	.	.	.	.	.	Ø	.	.	.	.	.	.	.	.	SEE TABLE A					
	.	.	.	.	.	.	Ø	.	.	.	.	.	.	.	SEE TABLE A					
	.	.	.	.	.	1	1	Ø	.	.	.	.	.	H	L	H	H	H	L	H
	.	.	.	.	.	1	1	1	.	.	.	.	H	H	H	L	H	H	L	H
$C000-$C01F	R	.	.	.	.	.	.	.	.	.	.	.	*	.	H	H	H	H	H	L
	W	.	.	.	.	.	.	.	.	.	.	.	.	.	H	H	H	H	H	H
$C020-$C0FF	.	.	.	.	.	.	.	.	.	.	.	.	.	.	H	H	H	H	H	H
$C100-$C2FF,	.	.	.	.	.	.	.	.	.	Ø	.	.	.	.	H	H	H	H	H	H
$C400-$C7FF	.	.	.	.	.	.	.	.	.	1	.	.	H	H	H	H	L	H	L	H
$C300-$C3FF	.	.	.	.	.	.	.	.	.	Ø	1	.	.	.	H	H	H	H	H	H
	.	.	.	.	.	.	.	.	.	1	.	.	H	H	H	H	L	H	L	H
	.	.	.	.	.	.	.	.	.	.	Ø	.	H	H	H	H	L	H	L	H
$C800-$CFFF	.	.	.	.	.	.	.	.	.	Ø	.	Ø	.	.	H	H	H	H	H	H
	.	.	.	.	.	.	.	.	.	1	.	.	H	H	H	H	L	H	L	H
	.	.	.	.	.	.	.	.	.	.	.	1	H	H	H	H	L	H	L	H
$D000-$DFFF	R	.	.	Ø	.	.	.	.	.	.	.	.	H	H	H	H	L	H	L	H
$E000-$FFFF	R	.	.	Ø	.	.	.	.	.	.	.	.	H	H	H	H	H	L	L	H
$D000-$FFFF	R	.	.	1	.	.	.	.	Ø	.	.	.	.	H	L	H	H	H	L	H
	R	.	.	1	.	.	.	.	1	.	.	.	H	H	H	L	H	H	L	H
	W	.	.	.	Ø	.	.	.	Ø	.	.	.	.	H	L	H	H	H	L	H
	W	.	.	.	Ø	.	.	.	1	.	.	.	H	H	H	L	H	H	L	H
**	.	.	.	.	.	.	.	.	.	.	.	.	.	.	H	H	H	H	L	H

LEGEND:
R - READ
W - WRITE
1 - SOFT SWITCH SET
Ø - SOFT SWITCH RESET
H - SIGNAL HIGH
L - SIGNAL LOW

NOTES
* KBD' is PHASE Ø gated. CXXX is not.
** All combinations not shown result in inactive condition (H H H H L H). Boldfaced signals are active.

Table 5.7 Activation of MD IN/OUT'.

ADDRESS RANGE	R/W'	DMA'	INH'	INT CXRM	SLOT C3RM	INT C8RM	PHS 0	MD IN/OUT'
$0000—$FFFF	W	L	-	-	-	-	H	H
$0000—$FFFF	R	-	L	-	-	-	H	H
$C020—$C0FF	R	-	-	-	-	-	H	H
$C100—$C2FF, $C400—$C7FF	R	-	-	0	-	-	H	H
$C300—$C3FF	R	-	-	0	1	-	H	H
$C800—$CFFF	R	-	-	0	-	0	H	H

LEGEND: R READ
W WRITE
1 SOFT SWITCH SET
0 SOFT SWITCH RESET
H SIGNAL HIGH
L SIGNAL LOW
- NO EFFECT

NOTE: All combinations not shown result in MD IN/OUT' low. Direction is into the data bus when MD IN/OUT' is high.

The order of important events in Figure 5.14 is:

1. RAS' rises near the end of PHASE 0. After IOU propagation delay, this enables the video ROW address to the multiplexed RAM address bus. The delay depends on the IOU, but the RAM ROW address must always be valid before RAS' falls.
2. PHASE 2 falls, clocking the data transfer of the previous 6502 machine cycle.
3. RAS' falls and clocks the video ROW address to RAM. RAS' low enables the video COLUMN address to the RAM address bus after IOU propagation delay. This delay must be short enough that the COLUMN address is valid before CAS' falls.
4. Early during PHASE 1, the 6502 address and R/W' become valid. The address will have no effect on RAM until the MMU takes control of the RAM address bus after RAS' rises. The R/W' line will have no effect on RAM until PHASE 0 rises because RAM R/W' is forced high during PHASE 1.
5. CAS' falls, clocking the video COLUMN address to RAM and initiating the transfer of video data from RAM to the data bus.
6. Video data becomes valid at the output of RAM. The time at which this occurs will depend upon the speed of the RAM chips installed in the Apple IIe. Assuming 200-nanosecond RAM is installed, the video data will become valid on the data bus no more than 135 nanoseconds after CAS' falls. This is well before PHASE 0 rises.
7. RAS' rises and enables the MPU ROW address to the RAM address bus after MMU propagation delay. This does not interfere with the previous RAM cycle because this new address is not clocked to RAM until RAS' falls.
8. PHASE 0 and CAS' rise simultaneously. PHASE 0 rising latches the video data in the video latch, and PHASE 0 high allows RAM R/W' to fall if this is an MPU write cycle. CAS' rising causes the RAM chips to bring their data output lines to high impedance after a delay. This delay will not exceed 50 nanoseconds if 200-nanosecond RAM chips are installed.
9. During PHASE 0, the MPU addresses RAM in the same way the video scanner did during PHASE 1, with RAS' selecting ROW or COLUMN, and RAS' and CAS' clocking the address to RAM. The only difference is that the MPU RAM address is developed in the MMU instead of the IOU. Just as in the scanner access, RAM data from the MPU access becomes valid on the data bus well before the data is transferred.
10. During write cycles, the 6502 write data must be valid before CAS' falls during PHASE 0. The write data stays valid until after PHASE 2 falls. The Apple write cycle is what is referred to in RAM literature as an early write cycle (as opposed to a read/write cycle). R/W' falls before CAS', and write data must be valid at the RAM input before CAS' falls.
11. PHASE 0 falls simultaneously with CAS' rising. If this is a write cycle, RAM R/W' begins to rise but takes a long time because the voltage is being pulled up by a 1K resistor. If this is a read

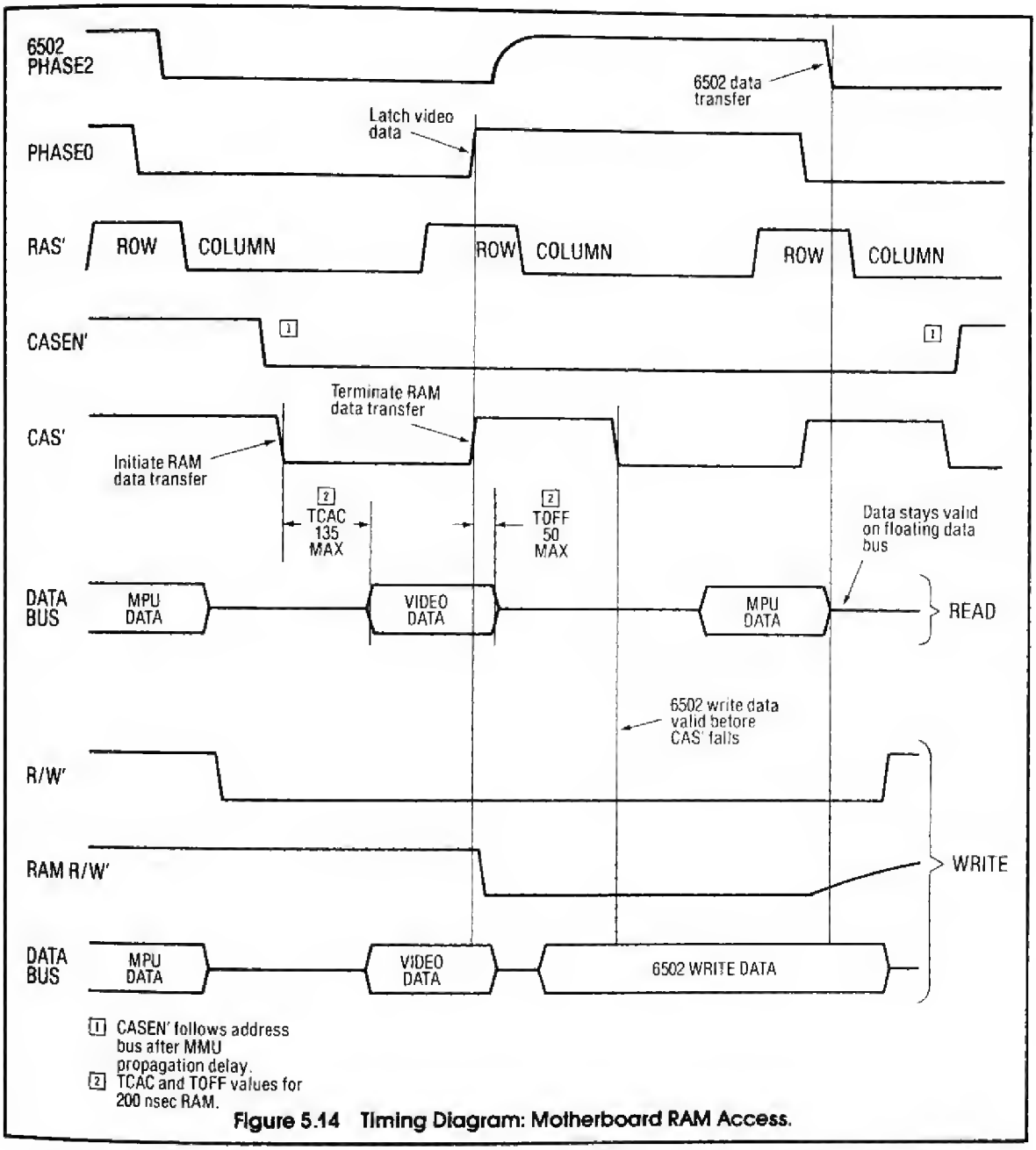


Figure 5.14 Timing Diagram: Motherboard RAM Access.

cycle, CAS' rising causes the RAM chips to bring their data outputs to high impedance after a delay of no more than 50 nanoseconds. The data bus may float before PHASE 2 falls, but the data from RAM will stay valid on the floating data bus.

12. RAM read data is clocked to the MPU by PHASE 2 falling.

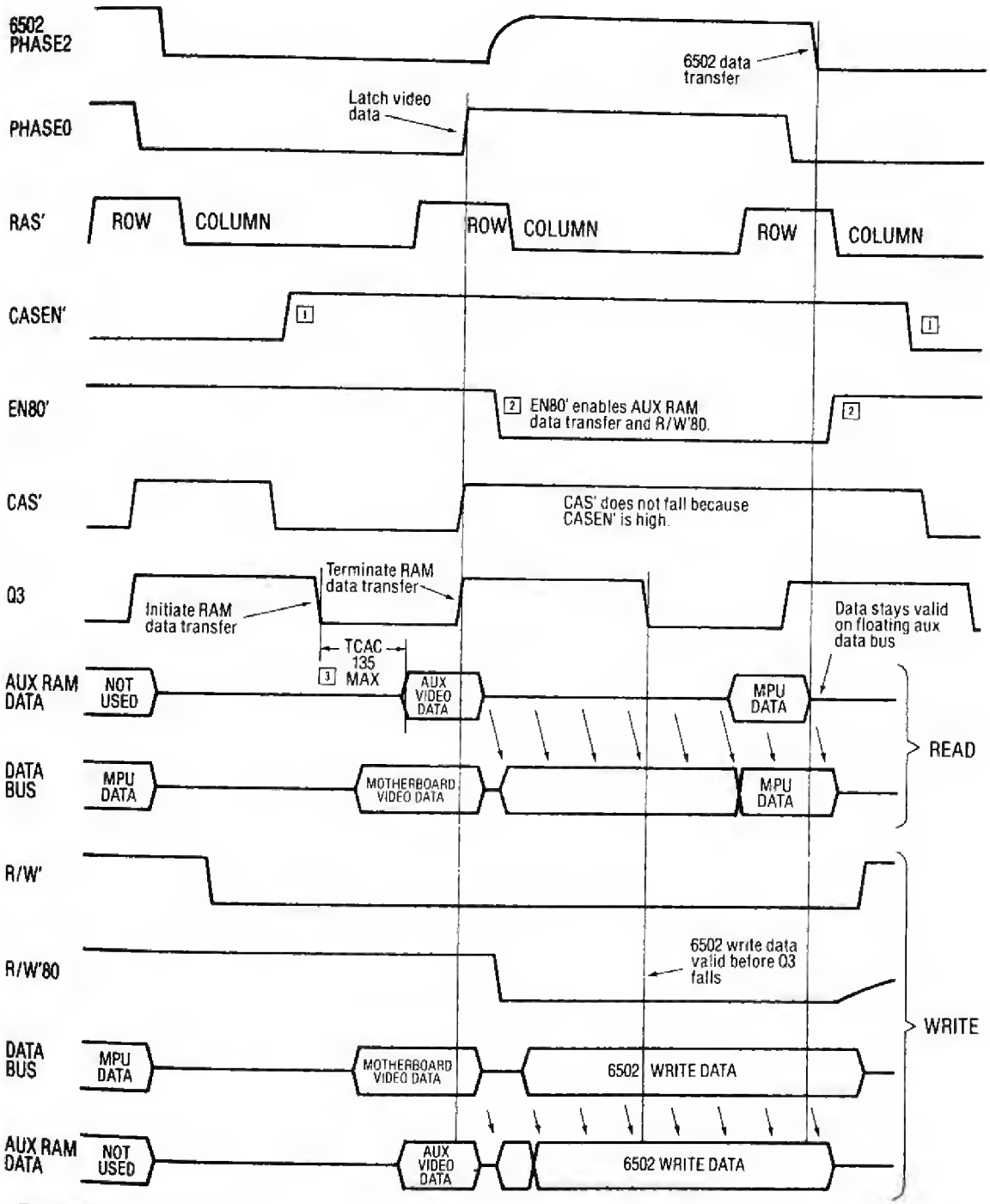
Figure 5.15 shows some basic details of video scanner read and 6502 read/write access to a 64K auxiliary RAM card. The response of auxiliary card RAM to its RAS' and CAS' inputs is identical to that of motherboard RAM, but timing details are different because Q3 is the CAS' input to the RAM chips on the auxiliary card, and because of the presence of the bidirectional data bus driver.

The development of the multiplexed RAM address is the same whether motherboard RAM or auxiliary card RAM is accessed. Motherboard RAM and auxiliary card RAM always have the identical address applied to their address inputs. The only difference in addressing detail is that CAS' falling clocks the COLUMN address to motherboard RAM while Q3 falling clocks the COLUMN address to auxiliary card RAM.

When an auxiliary card is installed, the video scanner simultaneously accesses the current scan address in motherboard and auxilary card RAM. Data from this access is saved in the motherboard and auxiliary card video data latches when PHASE 0 rises. Video scanner access to motherboard and auxiliary card RAM occurs every PHASE 1, regardless of which device the MPU accesses during PHASE 0.

The order of important events in Figure 5.15 is as follows:

1. PHASE 2 falls, clocking the data transfer of the previous 6502 cycle.
2. During PHASE 1, the video scanner ROW and COLUMN addresses are developed in the MMU as was described in the motherboard RAM timing section. RAS' falling clocks the video ROW address to auxiliary card RAM, and Q3 falling clocks the video COLUMN address to auxiliary card RAM.
3. The 6502 address and R/W' are set up early during PHASE 1. R/W' does not affect video scanner read access to auxiliary card RAM because R/W'80 is forced high during PHASE 1 (R/W'80 is gated by EN80' which is gated by PHASE 0 inside the MMU). Also, R/W' affects the direction of the auxiliary card bidirectional data bus driver, but this direction will not matter until EN80' falls and enables the bidirectional driver outputs. CASEN' from the MMU rises after the address is set up, but CAS' still falls since, in the timing HAL, CASEN' high only disables CAS' during PHASE 0. Therefore, MPU access to devices other than motherboard RAM does not interfere with video scanner access to motherboard RAM during PHASE 1.
4. Video data becomes valid on the auxiliary data bus no more than 135 nanoseconds after Q3 falls assuming 200-nanosecond RAM chips are installed. This is 70 nanoseconds later than motherboard RAM read data becomes valid, but data is still valid well before PHASE 0 rises. The auxiliary card video data is not passed to the data bus at this time because EN80' is high, disabling the auxiliary card bidirectional driver.
5. PHASE 0 and Q3 rise simultaneously. PHASE 0 rising clocks the video data to the auxiliary card and motherboard latches. The auxiliary card RAM data chips bring their data outputs to high impedance no more than 50 nanoseconds after Q3 rises if 200-nanosecond RAM is installed.
6. PHASE 0 high causes the MMU to bring EN80' low after propagation delay. EN80' in turn causes R/W'80, the auxiliary card RAM chip read/write control, to go low if this is an MPU write cycle. EN80' also enables the auxiliary card bidirectional driver to pass data between the auxiliary card and the data bus in the direction determined by R/W'. The immediate effect of this is to pass auxiliary data from the floating auxiliary data bus to the data bus in a read cycle, and to pass motherboard video data from the floating data bus to the auxiliary data bus during a write cycle.
7. During PHASE 0, no matter what Apple IIe device is being accessed, the address bus is multiplexed onto the RAM address bus by the MMU as described in the section on motherboard RAM timing. An access is made to auxiliary card RAM at this address during PHASE 0, even if the MPU is not accessing auxiliary card RAM. If the MPU is not accessing auxiliary card RAM, EN80' will remain high, forcing R/W'80 high and disabling the auxiliary card bidirectional driver. Since R/W'80 is high, the superfluous access to auxiliary card RAM is a read access, and the auxiliary card RAM is not modified, even in an MPU write cycle. Since the bidirectional driver is disabled, the data on the auxiliary data bus does not interfere with data bus communication between the MPU and other devices.
8. In a write cycle, 6502 write data becomes valid on the data bus early during PHASE 0. After the write data becomes valid at the data bus, it is propagated through the auxiliary card bidirectional driver to the RAM chips. This data must be valid at the auxiliary data bus before Q3 falls. Auxiliary card RAM timing is less critical than motherboard RAM timing in this regard because Q3 falls 70 nanoseconds after CAS' falls.



1 CASEN' follows address bus after MMU propagation delay.
2 EN80' follows PHASE0 after MMU propagation delay.
3 TCAC is 135 nsec MAX with 200 nsec RAM chips.

Figure 5.15 Timing Diagram: 64K Auxiliary Card RAM Access.

9. In a read cycle, the read data becomes valid on the auxiliary data bus no more than 135 nanoseconds after Q3 falls if 200-nanosecond RAM is installed. This data is propagated through the auxiliary card bidirectional driver to the data bus well before PHASE 2 falls.
10. PHASE 0 falls simultaneously with Q3 rising. The auxiliary card RAM chips bring their data output lines to high impedance no more than 50 nanoseconds after Q3 rises if 200-nanosecond RAM chips are installed. The floating auxiliary data bus will store the previous data for a long period of time so the RAM read data is passed through the bidirectional driver to the data bus, even though the auxiliary data bus is floating.
11. Auxiliary card RAM read data is clocked to the MPU by PHASE 2 falling. In a write cycle, the 6502 controls the data bus until shortly after PHASE 2 falls.
12. PHASE 0 low causes EN80' to rise after MMU propagation delay. This disables the auxiliary card bidirectional driver and, in a write cycle, causes R/W'80 to start rising. R/W'80 rises slowly because it is being pulled up by a 1K resistor.

THE 1K AUXILIARY RAM CARD

A good part of the circuitry of the MMU in the Apple IIe is devoted to management of 64K of motherboard RAM and 64K of auxiliary card RAM. Additionally, the 80-column capability is fully implemented in timing and firmware. So fully is the 64K auxilary RAM/80-column capability supported that one only has to install a simple card containing a latch, a bidirectional driver, and eight dynamic RAM chips in the auxiliary slot to achieve the capability.

Apple also developed another card for the auxiliary slot, one that supports the built-in 80-column capability but not the 64K auxiliary RAM capability. This 80-column only card contains a 1K static RAM and support circuitry. The 1K RAM is addressed by the MPU at $400—$7FF, and provides the display map for the alternate columns of an 80 x 24 text display. Figure 5.16 is a schematic which I drew from the board since I was unable to locate a schematic in any Apple literature.

The 1K RAM card is very similar to the 64K RAM card in data connection to an LS374 latch and to the data bus through an LS245 bidirectional driver. Control of the data transfer functions—identical to those of the 64K card—are as follows:

CONTROL SIGNAL	FUNCTION
RAS' falling	Latch ROW address
Q3 low	Save COLUMN address and enable RAM chip data transfer
R/W'80	RAM chip read/write control
PHASE 0 rising	Latch video data
PHASE 1 low	Enable aux data to video bus
R/W'	LS245 direction control
EN80'	LS245 enable

These control functions are dictated by the operation of motherboard circuitry. They enable the video scanner to drive the display map to the auxiliary latch during PHASE 1, and the MPU to transfer data to and from RAM during PHASE 0.

A very interesting feature of the 1K card is that it responds to EN80' without detecting the $400—$7FF range. For example, you could set RAMWRT' and store data at $8213, and the 1K RAM card would accept the data at the memory location normally addressed at $613. This should cause no harm since programs always check for the presence of a 64K card before attempting to store data in auxiliary card RAM outside of the $400—$7FF range. It does make it harder to test for a 64K card since either a 1K card or a 64K card will allow you to store and read back a value at any RAM address. The difference is that, when the installed card is a 1K card, you will read back the same value at any address that is separated from the modified address by a multiple of $400. The following programming example tests for a 64K card.

```
TEST64K STA $C001     SET 80STORE
        LDA $C057     SET HIRES
        LDA $C055     SET PAGE2
        LDA #$00
        STA $400      CLEAR $400
        LDA #$88      CTRL-H TEST VALUE
        STA $2000
        CMP $400      WAS $400 MODIFIED?
        BEQ NO64K     YES: NO 64K CARD
        CMP $2000     WAS $2000 MODIFIED?
        BNE NO64K     NO: NO 64K CARD
        CMP $2000     CHECK AGAIN
                      TO BE SURE*
        BEQ AUX64K    YES: 64K CARD FOUND
        BNE NO64K     NO: NO 64K CARD
```

*If an attempt to read auxiliary card RAM is made and no card is installed, the MPU will read the last value driven out of motherboard memory by the video scanner. $88 (left arrow) is not likely to be driven out of motherboard RAM if a normal text display is being scanned. Performing a double test for $88 precludes accidentally reading an $88 driven from the UNUSED 8 during HBL or VBL. All auxiliary card search programs should be cognizant of the possible contents of display memory.

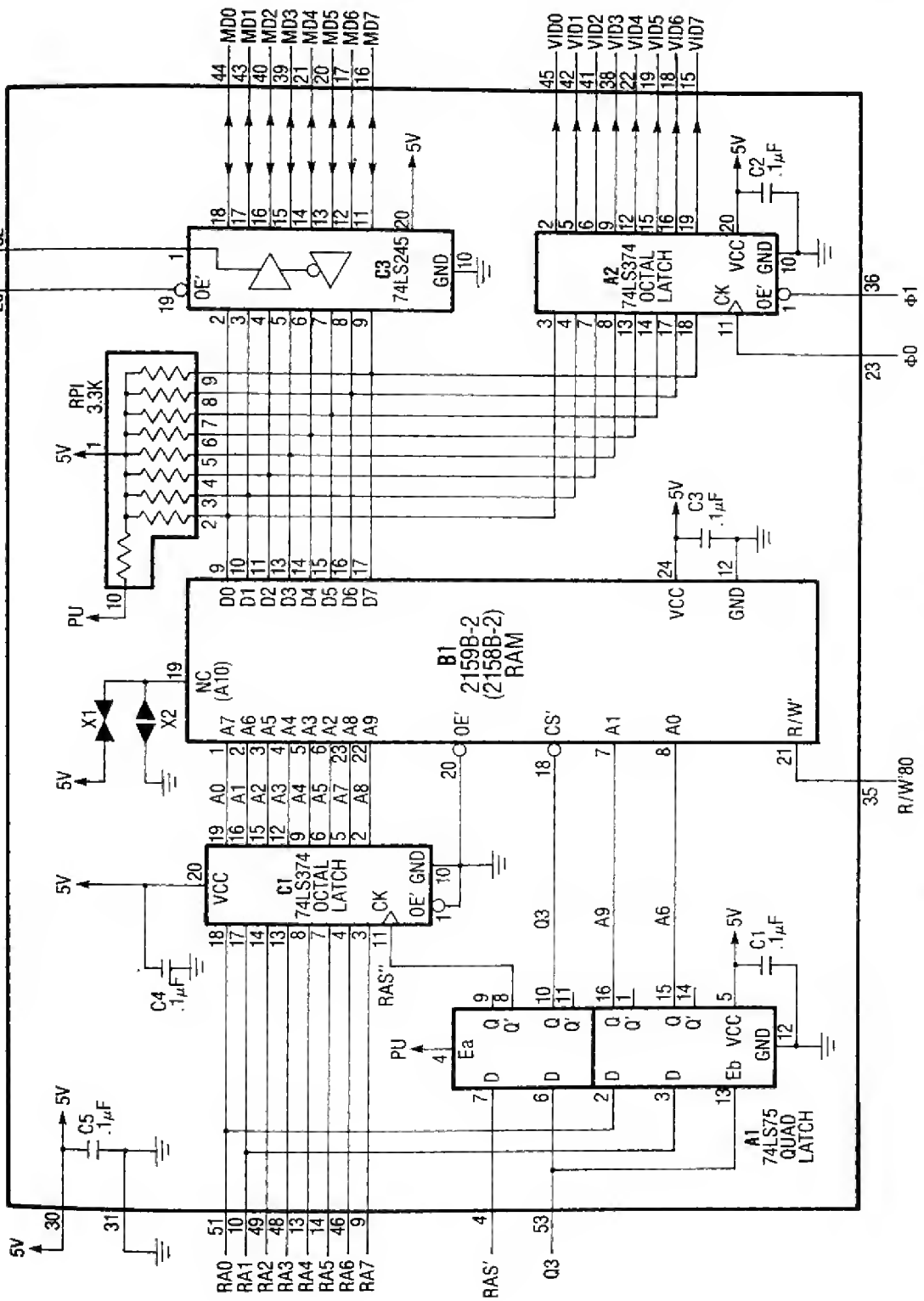


Figure 5.16 Schematic: The 1K Auxiliary RAM Card.

READING VIDEO DATA FROM A PROGRAM

In the October, 1982 issue of *Softtalk*magazine*, Bob Bishop showed that, by flagging display memory and reading the video scan data, a program could sync itself to the video scanner in the Apple II. I was excited by this capability and further documented it in *Understanding the Apple II*. It is also a capability of the Apple IIe, but is much less exciting because an Apple IIe program can easily sync itself to the video scanner by polling VBL' at $C019.

Most applications that can be accomplished with video polling can also be accomplished with a combination of VBL' polling and timed execution loops. The primary advantage of video polling is that a program can perform a substantial number of computations, then begin polling for a switching point. As a result, overall performance of a screen splitting program could sometimes be improved by using video polling rather than timed execution loops.

Even though there are conceivable advantages to video polling, the method is not commonly used in the Apple IIe. Yet the capabilty does exist so it is documented here. The purpose of this application note is to provide some reference material and discussion of programming techniques for those readers who wish to experiment with video polling.

The Apple II and IIe computers were not designed to allow a program to read the data driven out by the video scanner. It happens that it is possible because the data bus in the Apple II and IIe, and the peripheral data bus in the Apple IIe, store the previously valid data when they are floated. Figure 7.7 shows the timing of an Apple IIe read to a nonresponding address such as $C050 (GRAPHICS select). This figure shows that when such a read is made, the data bus and peripheral data bus take turns storing the video data while floating and transmitting the video data through the bidirectional driver to the other bus. It is surprising that the video data remains valid through the switching of the bidirectional driver, and in fact, data bus loading in the peripheral slots can make the video data impossible to read.**

*"Have an Apple Split", October 1982 *Softalk*, p 54.

**An exception to this is the case where auxiliary RAM is selected and read with no card installed in the auxiliary slot. This is a very reliable way to read video data, but relatively few Apples are operated with the auxiliary slot empty.

One reason for syncing a program to the video scan is to create displays that are a mixture of the normal Apple screen modes. For example, if you switch to HIRES before line 5 of every vertical scan and switch to LORES before line 10 of every vertical scan, you will have a stable combination of HIRES and LORES graphics displayed on the screen.

The method of reading video sync from a program is to set up flags in scanned memory at the point of the television scan where the program needs to take an action, such as switching from LORES to HIRES. Then the program polls a nonresponding address such as the cassette output port until it detects the flag. The choice of flags and their locations in memory will be dictated by the application.

Here are some addresses from which video data can be read:

$C02X	Cassette output
$C03X	Speaker output
$C04X	C040 STROBE'
$C05X	Screen switches and annunciator outputs
$C06X	Serial inputs (D0—D6 only)
$C07X	Timer trigger
$C08X	High RAM control
$C090—$C7FF	I/O control (use if slot is empty)
$CFFF	Expansion ROM disable

Of all of these choices, the screen switches stand out as having no chance of interfering with the operation of some device. The screen switches will normally be used, because the task of polling and switching can then be combined. Also, the polling loop is self documenting to investigators of the program. In the rare instance that one of the screen switches will not do for video polling, the annunciators, the C040 STROBE', and the cassette output addresses are likely choices.

Programming a combination display requires the normal programmer's imagination to conceive of the display. Additionally, a thorough grasp of memory scanning details is a necessity. Figure 5.17 is a diagram designed to aid the programmer in selecting locations for syncing flags. It should be used in conjunction with the memory maps from earlier sections of this chapter. From this diagram, one can quickly see the prospects for successful syncing at a given point in a given mode.

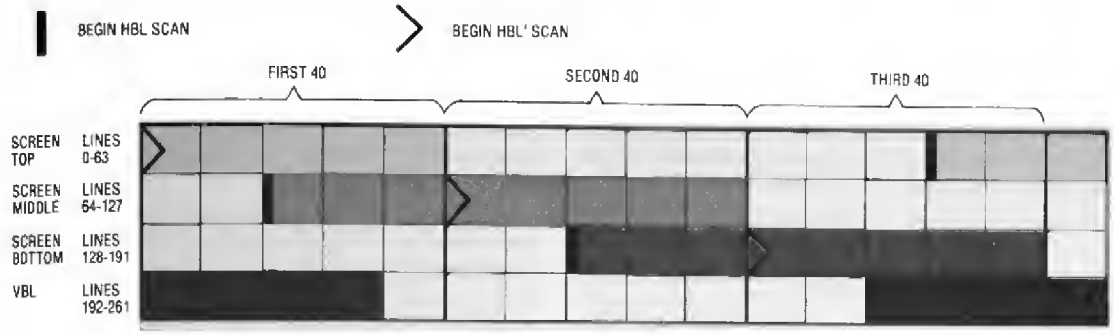


Figure 5.17 Display Memory Scanning.

A problem in all display modes is that most memory gets scanned more than once every vertical scan. Only the first 16 bytes of the SECOND 40 and the first 16 bytes of the THIRD 40 are scanned just once per vertical frame. This makes it more difficult to uniquely flag a scan position. An additional problem in TEXT/LORES is that the UNUSED 8 is used by DOS to store disk access information, so you often won't wish to set up polling flags there.

It is not necessary to uniquely flag a scan position. It is only necessary that there is no interference in detecting a scan flag between the present scan position and the flagged scan position. For example, assume we switched from HIRES to LORES during VBL and we wish to detect the middle of the screen. A byte of data stored at $5A8—$5CF is a detectable flag that results in horizontal lines across the screen at LORES positions 22 and 23. Even though parts of this memory are also scanned during HBL before positions 38 and 39, the next time they are scanned is during displayed position 22. There is no interference between the present location and the detection point.

The following is a typical video polling loop:

```
POLLIT  CMP $C050   FLAG VALUE IN ACCUM.
        BNE POLLIT
```

The loop takes seven clock pulses to execute and thus establishes one criteria for a screen flag. It must occupy a minimum of seven adjacent scanned bytes of memory or six bytes if one of the bytes is the first byte of a HBL scan. The first byte of a HBL scan is scanned twice, so six bytes here is the same as seven elsewhere.

Many display flagging techniques are possible. Here are some related ideas and facts:

1. Switching modes during HBL or VBL eliminates the unsightly display of switch points. Since VBL can easily be detected by polling $C019, most screen splitting schemes will utilize VBL as a reference.
2. The UNUSED 8 is the only undisplayed area available for flagging. It is scanned during HBL before the top third of the display and just after HBL during VBL. Use of the UNUSED 8 in flagging PAGE 1 of TEXT/LORES is restricted because of interference with the DOS.
3. Bit 7 of HIRES may be used as a flag and checked with the BIT instruction. This is one way of flagging the displayed HIRES area if bit 7 isn't critical for color or positioning.
4. When considering LORES flags in displayed areas, bits 0—3 control the upper block, and bits 4—7 control the lower block.
5. Video polling and VBL' polling work very well in conjunction with timed execution loops. There are 65 cycles in a horizontal scan—25 cycles of HBL and 40 cycles of HBL'. There are 262 horizontal scans in a vertical scan—64 in each third of the display screen and 70 during VBL. When a group of flagged bytes is located, it is then possible to find a precise byte in the group by slewing backwards in 17029-cycle loops until the first flagged byte is found. 17031-cycle loops can be used to slew forwards.
6. The first byte scanned during HBL is scanned twice in a row.
7. In TEXT/LORES, every memory line is scanned eight times in a row, except the last line of VBL which is scanned 14 times. In HIRES, every memory line is scanned once, but lines 250—255 are rescanned after line 255 (250 is the same as 256, 251 is same as 257, etc.).
8. Switching rapidly between GRAPHICS and TEXT mode will cause many televisions to lose color sync. This is a factor of alignment and response of the 3.58 MHz oscillator inside the television. Because of this unpredictability from

TV to TV, it is not possible to say what percentage of the time a program can leave the Apple in TEXT mode and still hope to maintain color sync. Commercial programmers could be well advised to keep at least a 50% GRAPHICS mode to TEXT mode ratio in their products if color stability is important.

9. Programs written for the Apple II may well not perform correctly on the Apple IIe because of differences in scanning during HBL. In the Apple II, HBL scanned memory was separate from other display memory in TEXT/LORES scanning. In the Apple IIe, HBL scanned memory overlaps other scanned memory in TEXT/LORES scanning in similar fashion to HIRES scanning. An example of one program that will not work in the Apple IIe is the UNDERLINE program from *Understanding the Apple II* (nuts!!).
10. Only motherboard (not auxiliary card) RAM video data is passed to the data bus, so only motherboard RAM need be flagged for video polling in DOUBLE-RES display modes.

Figure 5.18 is a programming example which demonstrates the video polling capability in the Apple IIe. It creates a split screen display with TEXT at the top and LORES graphics at the bottom. A frame of asterisks around the TEXT area enhances the display and provides a detectable flag which tells the program it is time to switch from TEXT to GRAPHICS. To demonstrate that processing can proceed while maintaining a split screen display, a message is scrolled accross a portion of the TEXT area, and horizontal lines are drawn in changing colors in the LORES area.

The scheme of the program is to switch to TEXT during VBL, and switch to GRAPHICS after detecting the bottom row of asterisks. The GRAPHICS display is updated during VBL and, if necessary, during the TEXT display period. The TEXT display is updated during the GRAPHICS display period. Since no area is modified while it is being scanned for display, the visual display changes without accompanying flicker.

The display updates performed by the demonstration program are admittedly simple, but there is time for more complex tasks. Since there are 17030 MPU cycles in a vertical scan, a lot can be accomplished during a scan portion. As shown by the demonstration program, operations such as drawing a LORES line or reading a paddle timer are easily performed within this time frame.

PROGRAM NOTES

Lines 55—58: After updating the LORES display, wait until the top row of asterisks is passed before starting to poll for asterisks. Otherwise, the program will detect the top row and switch to GRAPHICS at the top of the screen.

Lines 61—72: The last displayed 24 bytes of TEXT line 8 are scanned during HBL before TEXT line 0. Including the asterisks at the right of TEXT line 7 and the left of TEXT line 8, asterisk ASCII is driven out for 27 consecutive cycles before TEXT line 8. Therefore, the polling loop looks for a string of asterisks that exceeds 27 cycles.

Lines 73—74: Program flow arrives at line 73 at 56—62 cycles after the beginning of HBL before TEXT line 11. The desired switch point is during HBL before TEXT line 12, so the program must wait 464 (65 x 8 − 56) to 483 (65 x 8 − 62 + 25) cycles before switching to GRAPHICS.

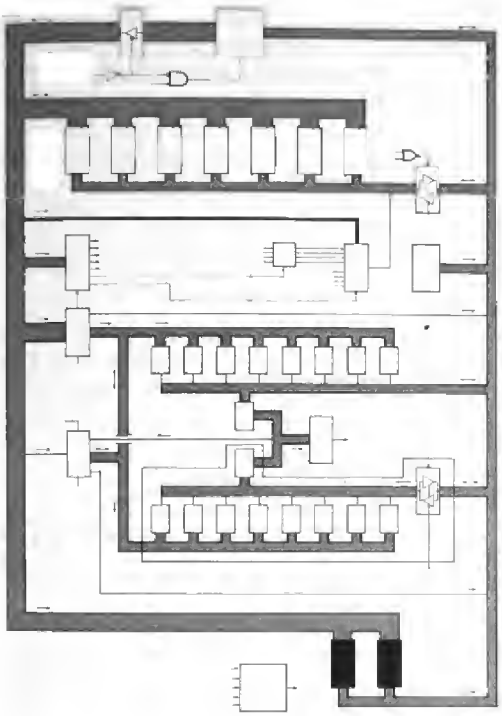
```
SOURCE FILE: VID POLL
                    1 ************************************************************
0000:               2 *
0000:               3 *                    VIDEO POLLING DEMO
0000:               4 *                      BY JIM SATHER
0000:               5 *                         4/17/84
0000:               6 *
                    7 ************************************************************
0006:               8 HLINV    EQU  $06       HLINE VERTICAL POSITION
0007:               9 PRNTCNT  EQU  $07       SCANS PER DISPAY SCROLL COUNTER
0008:              10 MSGX     EQU  $08       DISPLAY MESSAGE INDEX
0024:              11 CH       EQU  $24       COUT CURSOR HORIZONTAL POSITION
002C:              12 H2       EQU  $2C       RIGHT SIDE OF HLINE
C019:              13 VBLOFF   EQU  $C019      VBL' POLLING ADDRESS
C050:              14 GRAFIX   EQU  $C050      GRAPHICS ADDRESS
C051:              15 TEXT     EQU  $C051      TEXT ADDRESS
C052:              16 NOMIX    EQU  $C052      MIXED DISPLAY OFF ADDRESS
C056:              17 LORES    EQU  $C056      LORES ADDRESS
F819:              18 HLIN     EQU  $F819      DRAW HLINE SUBROUTINE
F85F:              19 NEXTCOL  EQU  $F85F      NEXT LORES COLOR SUBROUTINE
FB1E:              20 PREAD    EQU  $FB1E      READ PADDLE SUBROUTINE
FC58:              21 HOME     EQU  $FC58      HOME CURSOR SUBROUTINE
FCA8:              22 WAIT     EQU  $FCA8      WAIT BY ACCUMULATOR SUBROUTINE
FDED:              23 COUT     EQU  $FDED      CHARACTER OUTPUT SUBROUTINE
                   24 ************************************************************
----- NEXT OBJECT FILE NAME IS VID POLL.OBJ0
1000:              25          ORG  $1000
1000:AD 56 C0      26          LDA  LORES      INITIALIZE DISPLAY
1003:AD 52 C0      27          LDA  NOMIX
1006:A9 27         28          LDA  #39
1008:85 2C         29          STA  H2         HLINES END AT 39
100A:85 06         30          STA  HLINV      FIRST HLINE AT VERT = 39
100C:20 58 FC      31          JSR  HOME
100F:20 9C 10      32          JSR  PRNT40     DRAW AN ASTERISK FRAME
1012:A2 09         33          LDX  #9
1014:20 ED FD      34 LP9      JSR  COUT       FRAME IS 40X11
1017:A0 27         35          LDY  #39
1019:84 24         36          STY  CH
101B:20 ED FD      37          JSR  COUT
101E:CA            38          DEX
101F:D0 F3         39          BNE  LP9
1021:20 9C 10      40          JSR  PRNT40
                   41 ************************************************************
1024:AD 19 C0      42 VBLLP1   LDA  VBLOFF     POLL FOR VBL
1027:30 FB         43          BMI  VBLLP1
1029:AD 51 C0      44          LDA  TEXT       SWITCH TO TEXT
102C:20 5F F8      45          JSR  NEXTCOL    INCREMENT LORES COLOR
102F:A5 06         46          LDA  HLINV
1031:18            47          CLC
1032:69 01         48          ADC  #1         INCREMENT HLINE VERT COORDINATE
1034:C9 30         49          CMP  #48
1036:90 02         50          BCC  VERTOK
1038:A9 16         51          LDA  #22        HLINES DRAWN AT 22-47
103A:85 06         52 VERTOK   STA  HLINV
103C:A0 00         53          LDY  #0         START HLINE AT LEFT SIDE
103E:20 19 F8      54          JSR  HLIN       DRAW A LINE
1041:AD 19 C0      55 VBLLP2   LDA  VBLOFF     MAKE CERTAIN VBL IS GONE
1044:10 FB         56          BPL  VBLLP2
1046:A9 0C         57          LDA  #$C        WAIT 535 MPU CYCLES
1048:20 A8 FC      58          JSR  WAIT       BYPASS FIRST ROW OF ASTERISKS
                   59 ************************************************************
```

Figure 5.18 Assembler Listing: Video Polling Demonstration (1 of 2).

```
104B:A9 AA      60          LDA  #'*'      NOW FIND ASTERISK STRING
104D:CD 51 C0   61 ASTSKLP  CMP  TEXT
1050:D0 FB      62          BNE  ASTSKLP   2
1052:CD 51 C0   63          CMP  TEXT      6
1055:D0 F6      64          BNE  ASTSKLP   8
1057:CD 51 C0   65          CMP  TEXT      12
105A:D0 F1      66          BNE  ASTSKLP   14
105C:CD 51 C0   67          CMP  TEXT      18
105F:D0 EC      68          BNE  ASTSKLP   20
1061:CD 51 C0   69          CMP  TEXT      24
1064:D0 E7      70          BNE  ASTSKLP   26
1066:CD 51 C0   71          CMP  TEXT      30 THAT'S ENOUGH (GREATER THAN 27)
1069:D0 E2      72          BNE  ASTSKLP   32
106B:A9 0B      73          LDA  #$B       WAIT 464 MPU CYCLES
106D:20 A8 FC   74          JSR  WAIT      GOT "*"s; NOW WAIT TIL THEY PASS
                75 ************************************************************
1070:AD 50 C0   76          LDA  GRAFIX    SWITCH TO GRAPHICS MODE
1073:C6 07      77          DEC  PRNTCNT
1075:10 22      78          BPL  GOBACK
1077:A2 00      79          LDX  #0        READ PDL 0 TO GET DISPLAY SPEED
1079:20 1E FB   80          JSR  PREAD
107C:98         81          TYA
107D:29 1F      82          AND  #$1F      NOT TOO SLOW
107F:85 07      83          STA  PRNTCNT
1081:E6 08      84          INC  MSGX      TIME TO SCROLL MESSAGE
1083:A9 1F      85          LDA  #$1F
1085:25 08      86          AND  MSGX      WRAP MESSAGE INDEX AT $1F
1087:85 08      87          STA  MSGX
1089:A0 10      88          LDY  #16       DISPLAY WINDOW IS 16 LETTERS WIDE
108B:AA         89 DSPLYLP  TAX
108C:BD A7 10   90          LDA  MESSAGE,X
108F:99 8C 06   91          STA  $68C,Y    PRINT MESSAGE
1092:CA         92          DEX
1093:8A         93          TXA
1094:29 1F      94          AND  #$1F      WRAP AT $1F
1096:88         95          DEY
1097:10 F2      96          BPL  DSPLYLP
1099:4C 24 10   97 GOBACK   JMP  VBLLP1    GO WAIT FOR NEXT SCAN
                98 ************************************************************
109C:A2 28      99 PRNT40   LDX  #40       PRINT 40 ASTERISKS
109E:A9 AA     100          LDA  #'*'
10A0:20 ED FD  101 LP40     JSR  COUT
10A3:CA        102          DEX
10A4:D0 FA     103          BNE  LP40
10A6:60        104          RTS
               105 ************************************************************
10A7:D6 E9 E4  106 MESSAGE  ASC  'Video    Polling Demonstration -- '
10AA:E5 EF A0
10AD:D0 EF EC
10B0:EC E9 EE
10B3:E7 A0 C4
10B6:E5 ED EF
10B9:EE F3 F4
10BC:F2 E1 F4
10BF:E9 EF EE
10C2:A0 AD AD
10C5:AD A0
               107 ************************************************************

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Figure 5.18 Assembler Listing: Video Polling Demonstration (2 of 2).



chapter 6

ROM in the Apple IIe

Non-erasable, random access, read only memories have taken many forms in the history of digital computers. From vacuum tubes to diodes to small scale integration to large scale integration, there has always been a need for the general purpose computer to have a resident program ready to tell it what to do at turn on. With the coming of large read only memories on single chips, the scope of programs contained in ROM in general purpose computers has expanded greatly.

The Apple IIe design supports 16,128 bytes of motherboard ROM, addressed from $C100 through $FFFF. Additionally, several provisions exist for controlling the Apple via programs in ROM on peripheral cards. These include addressing peripheral ROM using a slot's assigned address area (its $C0nX DEVICE SELECT' range and its $CnXX I/O SELECT' range), addressing peripheral ROM using the I/O STROBE' ROM addresses ($C800—$CFFF), and inhibiting motherboard memory and stealing addresses $0000—$BFFF and $C100—$FFFF. All told, the Apple IIe computer has a very versatile capability for operating under control of programs stored in ROM.

The sophistication of the ROM chips makes their connections in the Apple simple and easy to understand. Nevertheless, the topic of ROM is involved enough to merit its own chapter. For the most part, discussion of ROM in this chapter is limited to the motherboard C1—DF and E0—FF ROMs, and peripheral slot ROMs containing 6502 machine code and related data. The keyboard ROM and video ROM are covered in Chapters 7 and 8 respectively.

ROM HARDWARE

The Apple IIe firmware is programmed into two **2365A type ROMs**. The 2365A is a 28-pin, 8192-byte, NMOS ROM with two programmable chip select inputs, an OE' input, an CE' input, and 200-, 300-, or 450-nanosecond access time. NMOS stands for Negative channel, Metal Oxide Semiconductor construction. These technical terms will help you if you look for the equivalent chip in a manufacturer's data book. Look under NMOS, 8192 x 8 ROMs.

A number of manufacturers make 28-pin, 8192-byte ROMs which would work in the Apple IIe. 2365A is the Synertek part number, and it is used

here because it is used in Apple schematics. It is reasonable to assume that 450-nanosecond ROMs are used since this is more than fast enough for Apple IIe timing.

Masked ROMs used in the Apple IIe are programmed by the manufacturer to the specifications of Apple Computer, Inc. Once the chip is built, only a casualty will cause alteration of the stored data. In addition to specifying the data to be contained in the ROM, Apple specifies which of the two **chip select** inputs are active high and which are active low. This is why the chip selects are said to be programmable. The chip selects of all ROMs on the Apple IIe motherboard—the C1—DF, E0—FF, video, and keyboard ROMs—are programmed so the INTEL type 2764, 2732, or 2716 EPROMs can be installed in their place.

C1—DF and E0—FF ROM connections in the Apple IIe are shown in Figure 6.1. The ROM chips are wired in the way most microcomputer hardware is wired, with repetitious, identical wiring going to both chips. It takes thirteen lines to address 8192 bytes, and the thirteen **address inputs** to the ROM chips are connected directly to A0—A12 of the address bus. The eight tri-state **data outputs** are connected directly to the data bus. These address and data connections are pleasantly simple when contrasted to RAM connections.

The only remaining connections are **power supply connections** (+5V and GROUND) and the **chip enabling inputs** (OE', CE', and the two chip selects). The chip enabling inputs have one thing in common in that, if any of them is inactive, the tri-state outputs of the ROM chip will be held at high impedance. More specifically, for the C1—DF or E0—FF ROM of the Apple IIe to control the data bus, its OE' and CE' inputs must be low, and its two CS inputs must be high. The CS inputs are both tied to +5 volts in the Apple IIe so there's not a lot to say about them. OE' and CE' are another story.

CE' is different than the other chip enabling inputs because, when it is high (inactive), the ROM chip goes in to a low current standby mode. Power supply loading can thus be reduced by inactivating CE' whenever ROM is not being accessed. Unfortunately, enabling the chip via CE' is slow (450 nsec max) compared to enabling a chip via OE' (150 nsec max), so OE' is used as the primary ROM chip enabling input in the Apple IIe. The CE' inputs to the ROM chips are used to mechanize the ROM disabling INHIBIT' and ENFIRM lines on Revision A motherboards. On Revision B motherboards, CE' is simply tied to ground.

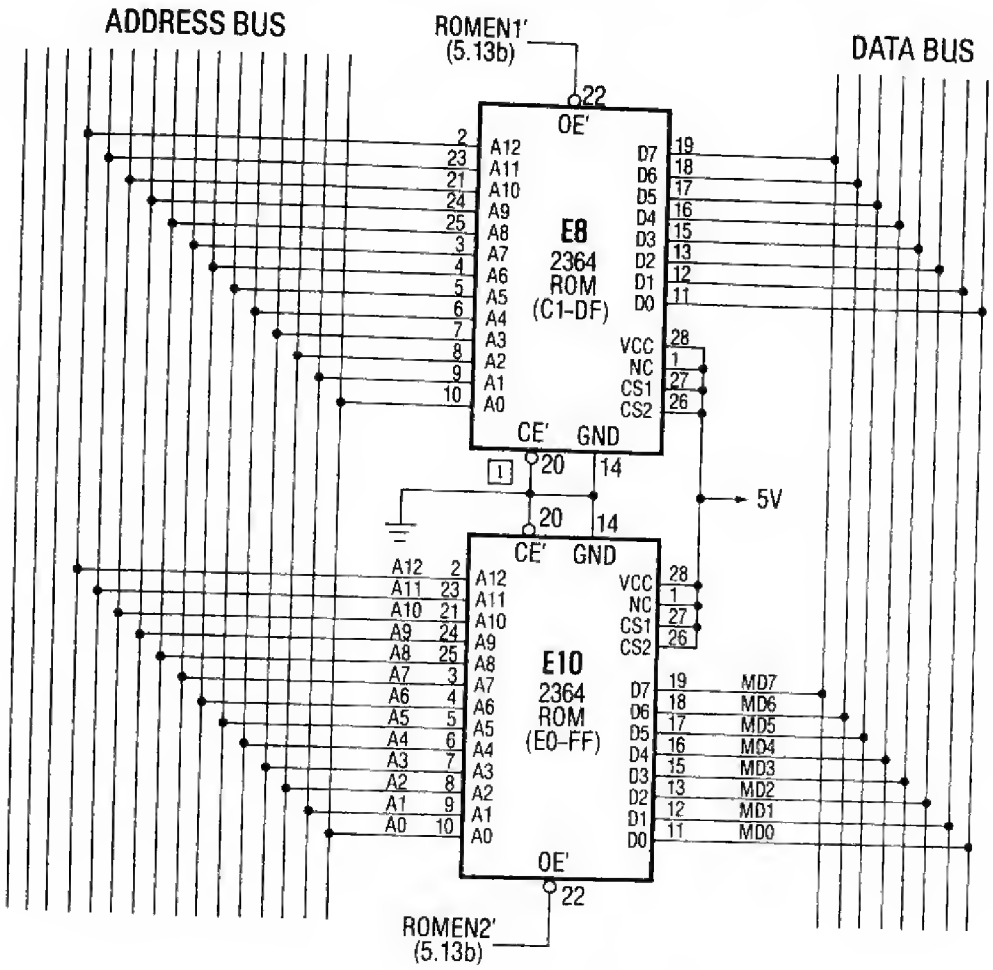
In Revision A, an auxiliary card can deactivate CE' to disable motherboard ROM by pulling ENFIRM low. I'm not sure what Apple had in mind for this capability, but they sacrificed it for DOUBLE-RES GRAPHICS in Revision B. The ENFIRM line was changed to FRCTXT', and the associated NAND gate is used to force text mode processing at the timing HAL by bringing gated GR+2' high when FRCTXT' is low (see Figure 3.9). This enables DOUBLE-RES GRAPHICS processing if 80COL is set and TEXT is reset. Additionally, the ROM disabling mechanization of INHIBIT' was transferred to the MMU. **In Revision B, all enabling and disabling of the C1—DF and E0—FF ROMs is mechanized via the ROMEN1' and ROMEN2' outputs from the MMU to the ROM OE' inputs.**

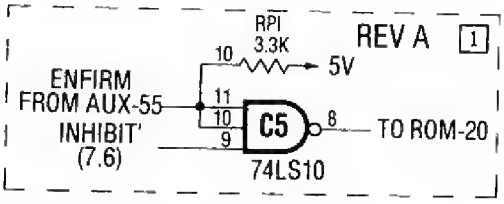
ROMEN1' AND ROMEN2'

ROMEN1' and ROMEN2' are the MMU data bus management signals which activate the C1—DF and E0—FF ROMs respectively. Their mechanization in the MMU and descriptions of how ROM fits into the Apple IIe memory map are contained in the memory management sections of Chapter 5, but features of memory management related to ROM are reiterated here.

Motherboard ROM is addressed at $C100—$FFFF, although $D000—$FFFF addressing is shared with high RAM, and $C100—$CFFF addressing is shared with the peripheral slots. $D000—$FFFF is thought of as the primary ROM range of the Apple IIe, and it is configured for ROM reading and high RAM writing any time RESET' falls. After a reset, a program can select ROM or high RAM for $D000—$FFFF read response by manipulating the HRAMRD soft switch as described in Chapter 5. When HRAMRD is reset and INHIBIT' is high, read access to $D000—$FFFF causes the MMU to bring ROMEN1' ($D000—$DFFF) or ROMEN2' ($E000—$FFFF) low during PHASE 0 (plus MMU propagation delay).

The $C100—$CFFF range of the Apple IIe is assigned primarily to the peripheral slots, but programs can disable slot response and enable ROM response by manipulating the INTCXROM, SLOTC3ROM, and INTC8ROM soft switches as described in Chapter 5. This occurs regularly since the 80-column firmware, Apple IIe diagnostics, and a number of monitor subroutines reside in the $C100—$CFFF range. When INHIBIT' is high and access is made to a $C100—$CFFF address that is configured for motherboard ROM response, the





1 In Revision A, ENFIRM and INHIBIT' were required to be high before ROM-20 (CE') would drop low. In Revision B, ENFIRM is replaced by FRCTXT', and INHIBIT' is a logic input to ROMEN1' and ROMEN2' in the MMU.

Figure 6.1 Schematic: ROM in the Apple IIe.

MMU brings ROMEN1′ low during PHASE 0 to enable transfer of data from the C1—DF ROM to the data bus.

Both ROMEN1′ and ROMEN2′ are gated by **INHIBIT′** from the peripheral slots.* This reveals some of the power of INHIBIT′. The function of ROM is to place data on the data bus when it is accessed. When INHIBIT′ is low, ROMEN1′ and ROMEN2′ cannot fall and ROM is disabled. It is as if the ROM were not installed. The peripheral card is then free to respond to $D000—$FFFF (and $C100—DFFF if it is configured for motherboard ROM response) in any desired manner.

While ROMEN1′ and ROMEN2′ responses to $D000—$FFFF are R/W′ gated, ROMEN1′ response to $C100—$CFFF is not. As a result, write access to an address in the $C100—$CFFF range that is configured for motherboard response causes the MPU to compete with the C1—DF ROM for control of the data bus. I'm not sure why Apple decided not to gate $C100—$CFFF ROMEN1′ response with R/W′, but I speculate that it was to achieve a misguided form of compatibility with the I/O SELECT′ and I/O STROBE′ signals. I/O SELECT′ and I/O STROBE′ respond to read and write access to $C100—$CFFF. Why not make ROMEN1′ the same? I mean, you never know when one of those programmers is going to want to write to motherboard ROM.

PERIPHERAL SLOT ROM

In addition to the C1—DF and E0—FF ROMs, a typical Apple IIe will have several peripheral card ROMs that contain programs which can be executed by the 6502. Generally, these will be I/O SELECT′ ROMs addressed at $CnXX, I/O STROBE′ ROMs addressed at $C800—$CFFF, or INHIBIT′ ROMs addressed anywhere in the $0000—$BFFF and $C100—$FFFF ranges. I/O SELECT′ and I/O STROBE′ ROMs respond to the I/O SELECT′ or I/O STROBE′ peripheral slot inputs, and normally contain programs which implement the I/O functions of a peripheral card. Peripheral card INHIBIT′ ROMs utilize the INHIBIT′ output of the peripheral slots to substitute peripheral card resident programs and data for motherboard resident programs and data.

An **I/O SELECT′ ROM** contains a 256-byte program which initializes I/O at a slot when BASIC/DOS PR#n or IN#n or monitor n CONTROL-P or n CONTROL-K commands are executed. An example is the bootstrap ROM of the Apple Disk II controller. The program in this ROM, accessed at $C6XX when the controller is installed in Slot 6, begins the boot procedure that causes the DOS image on a diskette to be transferred to motherboard RAM. The 256-byte program is not nearly big enough to accomplish this task, but it is big enough to start the task and transfer some code from the diskette to RAM so the task can be continued.

There are actually very few I/O tasks that can be accomplished with a 256-byte driver. For this reason, an I/O SELECT′ ROM will often be supplemented by a 2K driver on an **I/O STROBE′ ROM**. The I/O STROBE′ signal goes low at the peripheral slots any time access is made to $C800—CFFF when INTCXROM and INTC8ROM are reset. Under a protocol described in Chapter 7, any peripheral card—but only one at a time—can activate response to the I/O STROBE′. The I/O STROBE′ could be utilized to enable any sort of peripheral card device, but it is almost always used to enable a 2K ROM containing some sort of I/O driver.

The third type of peripheral slot ROM is utilized in conjunction with the **INHIBIT′ line** which is a peripheral slot output, as opposed to the I/O SELECT′ and I/O STROBE′ inputs. Whereas the I/O SELECT′ and I/O STROBE′ outputs are ROM enabling signals that allow the MPU to access peripheral card ROM, the INHIBIT′ output is a memory disabling signal that allows peripheral cards to selectively disable motherboard and auxiliary card memory in favor of peripheral card response to MPU addressing.

An INHIBIT′ based peripheral does not have to substitute ROM for motherboard memory, but it certainly can. An example of a peripheral that does so is Apple's 12K firmware card which steals $D000—$FFFF from motherboard memory and substitutes peripheral card ROM or EPROM. Through the 12K firmware card, an Apple IIe user can have the convenience of both Applesoft and Integer BASIC in ROM.

ROM TIMING

ROM read timing is very simple and is the same for read access to the C1—DF, E0—FF, or keyboard ROMs. Among the points to consider are the timing characteristics of the ROM chips, disabling of motherboard and auxiliary card RAM via CASEN′ and EN80′, and enabling of the addressed ROM via

*As described in Chapter 5, CASEN′ and EN80′ are also gated by INHIBIT′ so MPU communication with RAM, as well as ROM, is disabled when INHIBIT′ is low.

ROMEN1', ROMEN2', or KBD'. Assuming Apple uses 450-nanosecond ROM, important timing characteristics are as follows:

1. The output data will become valid a maximum of 450 nanoseconds after the address input becomes valid.
2. The output data will become valid a maximum of 150 nanoseconds after the OE' input falls.*
3. The data output will go to high impedance a maximum of 150 nanoseconds after the OE' input rises.*

ROM read timing in the Apple IIe is illustrated in Figure 6.2. The main order of events is:

1. During PHASE 1, the video scanner performs its normal access to RAM.
2. At approximately 100 nanoseconds after 6502 PHASE 2 falls, the MPU address becomes valid. ROM data will be valid no more than 450

*Values given are for Synertek SY2365 ROM. Some ROM manufacturers have shorter OE' access specifications for 450 nanosecond ROM.

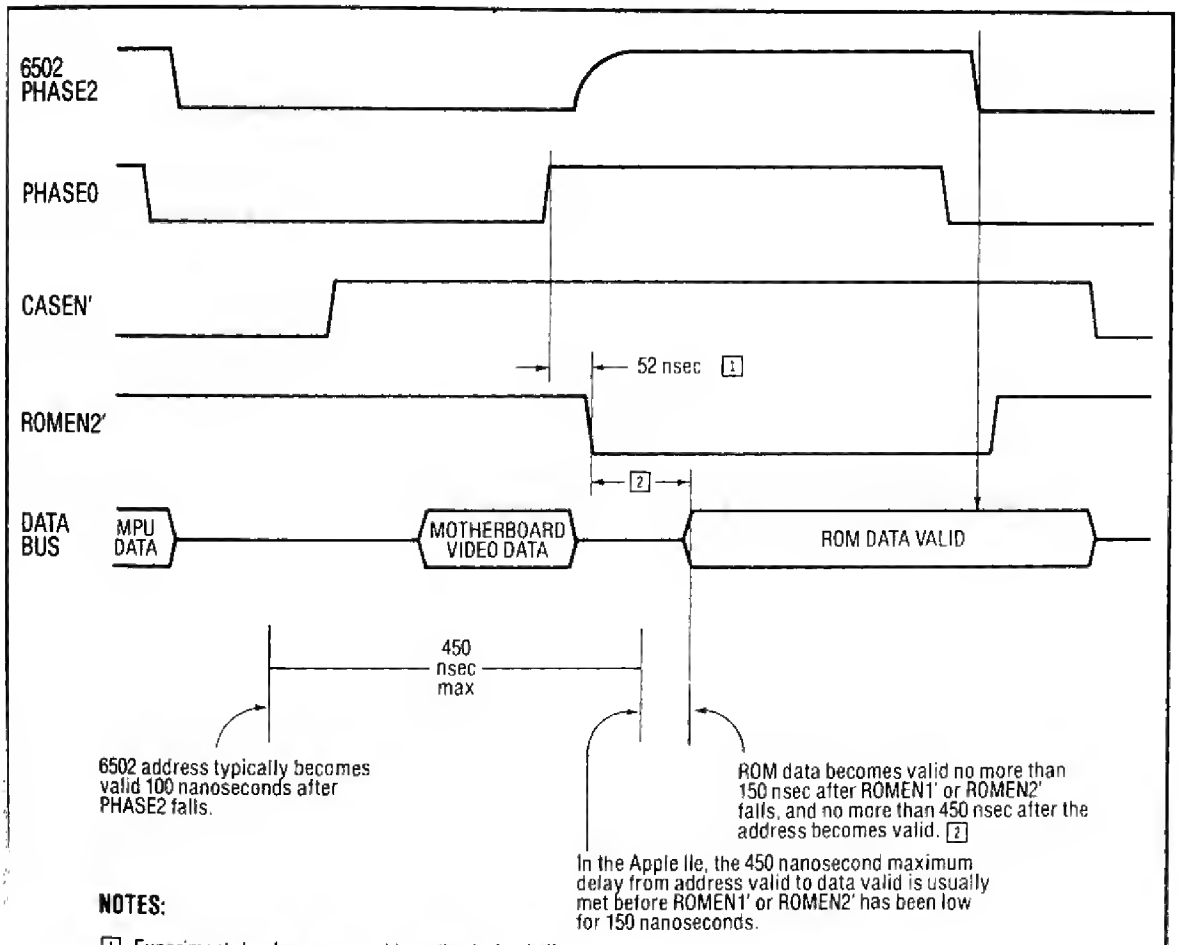


Figure 6.2 Timing Diagram: Read Access to the E0—FF ROM.

nanoseconds after this point if the ROM OE' input has been low sufficiently long.

3. If access in the previous cycle was to motherboard RAM, CASEN' rises when the ROM address becomes valid plus MMU propagation delay. This will disable MPU communication with motherboard RAM during PHASE 0.
4. PHASE 0 rises, causing ROMEN1', ROMEN2', or KBD' to fall and EN80' to rise (if previous cycle access was to auxiliary card RAM) after MMU propagation delay. CAS' and Q3 also rise when PHASE 0 rises, terminating the video scanner access to RAM. The data bus then floats, storing motherboard video data until the ROM data becomes valid.
5. 150 nanoseconds maximum after the ROM enabling signal falls, the ROM data becomes valid, assuming the address bus became valid early enough. With a typical 6502A and ROM chip, the address becomes valid early enough that the OE' to data valid delay of the ROM chip will determine when data becomes valid. In any case, ROM data becomes valid well before 6502 PHASE 2 falls.
6. PHASE 0 falls, followed by 6502 PHASE 2 falling, followed by ROMEN1', ROMEN2', or KBD' rising (if next cycle access is not to the same ROM chip), and EN80' falling (if next cycle access is to auxiliary card RAM). MMU propagation delay is longer than 6502 PHASE 0 to PHASE 2 propagation delay in my Apple IIe and perhaps in all Apple IIe's. ROM will control the data bus for a maximum of 150 nanoseconds after its OE' input rises.
7. If next cycle access is to motherboard RAM, CASEN' will fall when the next address becomes valid plus MMU propagation delay.

FIRMWARE IN THE APPLE

The hard features of the Apple ultimately determine its capabilities and limitations. However, the computer is only as powerful as the program controlling it at any given moment. Possibly the most important programs ever written for the Apple are the ones stored in the motherboard ROM. These give life to the machine and are used so often that we forget that they are just programs and that operational features created by any program can be changed.

The contents of Apple II and Apple IIe firmware have been the subject of many writings which this book cannot hope to match in a limited space. The goal here is just to give an overview of the Apple firmware and to provide some insight into the importance of the programs contained there. The approach is historical, beginning with the Apple II and progressing to the recently released firmware upgrades to the Apple IIe.

First, there was Integer BASIC, the monitor ROM, and some 6502 utilities. These programs were written primarily by Steve Wozniak, the designer of the original Apple II computer. This was in the bad old days before the proliferation of inexpensive disk drives, and before Microsoft Inc. started supplying all the computer companies with more sophisticated BASIC interpreters. Integer BASIC takes up about 5K of memory and has some very important limitations: no floating point arithmetic, no HIRES graphics commands, and no subscripted string variables, to name three. This is not said to belittle the considerable design accomplishments of Mr. Wozniak. It's just that within a few years, Integer BASIC became less than state-of-the-art.

The 6502 utilities included with Integer BASIC were some floating point arithmetic routines, the "SWEET 16" double word length command interpreter, and most importantly, the Mini-Assembler. **SWEET 16** is a small but sophisticated computer language which lets the programmer manipulate data in 16-bit word lengths. It utilizes RAM addresses $0 through $1F as 16 registers of 16 bits each, and has normal machine language commands such as ADD, SUBTRACT, COMPARE, BRANCH, etc. Writing some programs in SWEET 16 will make them take less space than the equivalent 6502 program, but the 6502 program will run faster.

The **Mini-Assembler** is the utility on which uncounted numbers of Apple owners have first learned to program 6502 assembly language. Its use is fully described in the *Apple II Reference Manual for IIe Only*.

SWEET 16 and the floating point routines are not described in any published Apple literature. In the old red Apple II reference manual, there are source/object listings of SWEET 16, the floating point routines, and the Mini-Assembler. SWEET 16 is fully described in the November, 1977 edition of *Byte* magazine ("SWEET 16: the 6502 Dream Machine" by Steve Wozniak).

The System Monitor

In microcomputer terminology, a system monitor is a program containing the most basic utilities of the system. Before the innovation of BASIC in ROM, a **monitor** in ROM was the primary user interface to the microcomputer. Some basic routines of a system monitor are keyboard input routines, video output

routines, memory display and modification routines, and storage media input/output routines.

The Apple was one of the microcomputers which led the transition from a monitor in ROM to BASIC in ROM as the primary human to machine interface for home computers. However, the Apple does have an extensive monitor in ROM, and the older Apples came up in the system monitor, not in BASIC.

The original Apple II system monitor contained such important Apple utilities as entry to BASIC, keyboard input, video text and LORES graphics output, cassette I/O, assignment of different peripheral slots as primary input or output, memory display and modification, a 6502 disassembler, machine language single step, trace, or normal subroutine execution, handlers for RESET', NMI', IRQ', and BREAK, and some 16-bit multiply and divide routines. Control of the monitor is via a highly usable command interpreter, the use of which is well described in the reference manual.

This, then, was the Apple: a cassette based system with a poor man's BASIC and a rich man's monitor in ROM and two empty 2K ROM sockets for user firmware. Oh yes, one more important thing—Apple published a source/object listing of the system monitor. This was a risky move which paid off immeasurably. By publishing their listing, Apple opened the way for investigators to learn thoroughly the nuts and bolts operation of the Apple. They thus aided competitors in developing numerous software and hardware applications for the Apple II. With this combination of extensive available applications and freedom of information, Apple pulled off the delicate trick of appealing not only to the masses who don't care how it works but also to more serious users who develop even more applications.

The system monitor dictates many of the operational characteristics of the Apple: what happens when the computer is turned on or RESET is pressed, the format of the screen text, the nature of cursor moves, and the assignment of primary input and output devices. The monitor zero page assignments must be taken into account by software designers. Page 2 of RAM is thought of as the Apple keyboard input buffer because that is the way the monitor uses Page 2. Cards capable of being primary input or output devices must have programs at their I/O SELECT' addresses, because the firmware assigns a slot as a primary input or output by jumping to the slot's first I/O SELECT' address ($C100 for Slot 1, $C200 for Slot 2, etc.). These features are not inviolate in the Apple. They can be changed by loading a new operating system into RAM or by replacing a single ROM chip, which is exactly what Apple did when they decided to change a few operational features of their computer.

The Apple II Plus

The Apple II evolved rapidly in the late 1970s into a more sophisticated machine than was originally introduced. The dynamic nature of the hardware and software support provided to the Apple II by Apple in 1978 and 1979 is remarkable. In approximate order,

1. Applesoft BASIC became available on cassette,
2. The Disk II was introduced with its powerful DOS,
3. Applesoft became available on diskette,
4. Applesoft became available on a firmware card,
5. The Apple II Plus was released with Applesoft and Autostart monitor in ROM,
6. The Language System was released with 16K RAM card, Pascal language, and 16-sector disk capability, and
7. DOS 3.3 was released with its 16-sector capability (August 1980).

A popular Apple configuration became both Integer BASIC and Applesoft in ROM with automatic selection between the two by the DOS when a program was RUN. One BASIC resided in motherboard ROM, and the other resided in ROM on a Slot 0 **firmware card**. As this configuration demonstrates, bank switching of firmware operating systems is a very powerful concept.

The availability of **Applesoft BASIC** greatly improved the versatility of the Apple by making the manipulation of the HIRES screen, large disk text files, and floating point numbers practical. Unfortunate Applesoft weaknesses were incompatibility in command and memory usage with Integer BASIC, no AUTO numbering, no DSP (DiSPlay) command, and the absence of the 6502 Mini-Assembler. Also, the SWEET 16 interpreter and old floating point routines which are associated with Integer BASIC are not available with Applesoft.

The new **Autostart** monitor reflected the changing nature of the personal computer owner. It caused the Apple to come up in BASIC instead of the system monitor, gave the Apple the capability to boot a disk at power up, and greatly improved the ESCape mode cursor moves. In the process, the SINGLE STEP and TRACE investigative utilities for machine language programs and the 16-bit multiply and divide routines were removed. Programmers' utilities were thus sacrificed for improved operational features. The small businessmen gained a system that automatically loads and starts at

power up, and the computer hacks lost the convenience of STEP and TRACE in ROM.

The Impact of the RAM Card

An eventual development in the evolution of the Apple II was the popularity of the **16K RAM card.** With a disk based system, as the Apple had become, it was no longer as necessary to have extensive operating systems in ROM as it was with a cassette based system. The entire Integer BASIC program and associated utilities could be loaded into the $E000—$F7FF area of the 16K card in a tolerably short period of time, giving the user the equivalent of the firmware card but more versatility. The system monitor became alterable by the user, and the number of operating systems that might possibly reside in high memory became unlimited. The disadvantages of the 16K RAM card were the possibility of overwriting high memory, occasional extra waiting for loading the program into the 16K RAM card from disk, the impossibility of protecting 6502 vectors from program encrypting artists, and the lost capability of other peripheral cards to respond to $F800—$FFFF addressing. For most users, the advantages of the 16K RAM card outweighed the disadvantages.

The Apple IIe

In early 1983, Apple released the Apple IIe computer, an improved Apple II with increased firmware requirements. The Apple IIe is a 128K, upper/lower case, 80-column computer designed to emulate a 48K Apple II with 16K RAM card in Slot 0 and 80-column display card in Slot 3. The 16K RAM card emulation is strictly a hardware mechanization with 64K of motherboard RAM, MMU soft switches controlled by $C08X access, and data bus management via the ROMEN1′, ROMEN2′, CASEN′, and EN80′ MMU outputs. Two very important operational improvements over the 16K RAM card are that high RAM is disabled when INHIBIT′ is low, and that high RAM is disabled for reading and enabled for writing any time RESET′ falls.

The 80-column card emulation is also a hardware mechanization and is implemented in the timing generator, auxiliary RAM card, video generator, and MMU. Additionally, extensive new firmware is utilized to accomplish 80-column text output.

The 80-column firmware of the Apple IIe coresides, in ROM, with the old 40-column firmware of the Apple II. The 40-column firmware is enabled at power up or system reset, and the 80-column firmware must be selected via IN#3 or PR#3. The 80-column firmware accomplishes the task of maintaining the alternating column auxiliary card and motherboard RAM text display map. It also performs some Pascal housekeeping and interprets special control and escape characters that are used to manipulate the text display. Additionally, the XFER (transfer control between motherboard and auxiliary card RAM) and AUXMOVE (transfer data between motherboard and auxiliary card RAM) utilities reside in the $C3XX range and are considered to be part of the 80-column firmware.

To make room for the firmware necessities of the Apple IIe, Apple switched from 12K of ROM in the Apple II to 16K of ROM in the Apple IIe. They could have accomplished the same thing by bank switching the $D000—DFFF range with ROM as is done with high RAM, but they chose, instead, to switch the $C100—$CFFF between I/O and ROM. This is consistent with the concept of an emulated Slot 3 80-column card. Slot 3 firmware is accessed at $C3XX and $C800—$CFFF, and so is the Apple IIe 80-column firmware.

Other than the presence of 80-column firmware, the monitor in the Apple IIe is nearly identical to the Apple II Autostart monitor. There are some modifications, such as a new KEYIN routine, interpretation of ESCAPE arrow cursor moves, and interpretation of open and close Apple keys at reset time. The modifications increase the size of the monitor and necessitate the location of some monitor routines in the $C100—$C2FF addressing range. These routines are called by executing GOTOCX at $FBB4 with the index of the desired routine in the Y-register.

Another firmware addition made in the Apple IIe is the built-in diagnostics at $C400—$C7FF. This is the only portion of the $C100—$CFFF firmware for which Apple did not publish a listing. The diagnostics include checks of the ability to control and read IOU and MMU soft switches, checksum computations for the C1—DF and E0—FF ROMs, and a complete check of motherboard (but not auxiliary card) RAM.

The bottom 256 bytes of the C1—DF ROM cannot be read by the MPU since ROM is not addressed at $C0XX. I put the ROM in my PROM burner to see what Apple puts in the this unaccessed portion. The answer? All zeroes.

The Apple IIe Firmware Upgrade

When Apple developed the Apple IIe, they had to struggle with the sometimes conflicting goals of building the best computer possible and retaining maximum compatibility with the Apple II. Some of

the resulting compromises were less than perfect. My primary dislikes are the treatment of the 80-column capability as a Slot 3 peripheral, the failure to upgrade Applesoft commands to support DOUBLE-RES GRAPHICS, and the failure to upgrade BASIC and monitor command interpretation subroutines so that lower case keyboard input can be interpreted as commands.

While developing the Apple IIc, Apple eliminated this last imperfection and made some other changes to the firmware. These changes support Apple IIc features including built-in mouse capability, general I/O structure, and 65C02 MPU. The firmware improvements in the Apple IIc program have spawned a firmware upgrade to the Apple IIe. This upgrade is expected to be released at approximately the same time as this book. Based on a preliminary version that Apple supplied, some features of the firmware upgrade are briefly described here.*

The goals of the firmware upgrade are to maximize Apple IIe/IIc compatibility, to integrate displayed icons into the Apple IIe display, and to make general improvements to Apple IIe firmware. Included in the package are new C1—DF and E0—FF ROMs, a new video ROM, and a 65C02 MPU.

Icons are pictures that represent things, and Apple today is a champion of the concept of representing computer actions with icons on the screen and selecting the action by "pointing" at it with a mouse. In line with this company policy, Apple has developed a video ROM with icon patterns replacing the INVERSE upper case patterns at addresses $200—$2FF (see Figure 8.8). The video ROM with "mouse text" is installed in all Apple IIc's, and it is included in the firmware upgrade to the Apple IIe. There is still an INVERSE upper case set at $000—$0FF of the new video ROM, so this capability is not lost. However, INVERSE video output routines which output INVERSE upper case text characters by storing $40—$5F in the display map and setting ALTCHRSET will display mouse icons instead of the desired text characters with the mousy video ROM.

In addition to the "mouse text" in the video ROM, there are improvements to the system monitor, the 80-column firmware, and Applesoft in the firmware upgrade. These include a new IRQ'/BREAK handler, a 6502 miniassembler, an ASCII (in addition to hexadecimal) monitor input mode, a monitor SEARCH command, faster and smoother 80-column firmware text scrolling, and monitor, Applesoft, Pascal, and ProDOS interpretation of lowercase commands. To make room for these improvements, various routines in the $CXXX and $F775—$FFFF ranges are streamlined, relocated, or eliminated. Most notably, the firmware diagnostics are reduced in length by nearly two memory pages.

The new firmware upgrade looks like a winner. Substantial operational improvements are made to the Apple IIe, and the user gives up little in return. My suspicion is that the new firmware and 65C02 MPU will quickly make their way into the majority of Apple IIe's and that the characteristics of the enhanced firmware will come to represent the operational characteristics of the Apple IIe.

*Also see the NMI' and IRQ' and 65C02 MICROPROCESSOR sections of Chapter 4.

HARDWARE/SOFTWARE APPLICATION

MODIFYING THE SYSTEM MONITOR

The system monitor determines many of the operational features of the Apple IIe, and modifying the system monitor is one way of enhancing the Apple IIe. The monitor resides in the $F800 to $FFFF space of the E0—FF ROM, and the $C100—$C2FF space of the C1—DF ROM.* It is easily modified in high RAM by the user and by commercial programs, so modifying the monitor in RAM will give you access to your personal utilities but won't necessarily help you investigate programs written by others. If you want an unalterable modification, you must make it in EPROM and have your Apple configured so that you can override software control to enable your EPROM.

The system monitor is a fine program with some good utilities that you probably don't want to delete. However, addresses $FCC9—$FD0B, $FECD—$FEF5, and $FEFD—$FF2C are cassette read/write routines that are rarely used in a disk based Apple IIe. Modify the cassette routines to run at a RAM address and save them on diskette. This generates 156 bytes for your personal firmware which can be increased to 164 by moving the control-Y JMP statement to $FEC2. This is accomplished by storing $4C, $F8, and $03 at $FEC2—$FEC4 and by storing $C1 at $FFE4.

Other areas of interest to monitor modifiers are in the 80-column firmware and the firmware diagnostic routines. There are 13 consecutive unused bytes at $C3F3—C3FF, four consecutive unused bytes at $C9A6—$C9A9, and four consecutive unused bytes at $C7FC—$C7FF. These locations are convenient areas for making minor patches to the GOTOCX subroutines.

If you need more room for a firmware application than provided by the above suggestions, you can make room for 1024 continuous bytes of code by eliminating the firmware diagnostics. What's more, you can execute any program that begins at $C401 by holding down close Apple and pressing CONTROL-RESET. Programs beginning at other locations in the $C400—$C7FF range can be executed by setting INTCXROM and calling them via a JMP or JSR instruction. Replacing the diagnostics with your firmware applications doesn't mean that the Apple IIe self-diagnostic capability is permanently lost. It only means you would have to install the standard Apple C1—DF ROM to run the firmware diagnostics.

The idea in making a minor modification to the system monitor is to create an EPROM with contents identical to the monitor except for small areas which contain your data. One idea for modification is to increase the command repertoire of the monitor. STEP and TRACE routines were deleted when the Autostart monitor was developed. This opened space for two commands in the command tables (CHRTBL and SUBTBL) of the monitor. If you delete the cassette routines, that will also open up two more command spaces since the READ and WRITE commands will no longer function.

What sort of commands can be installed? Here are some possibilities:

1. Breakpoint insert and breakpoint remove commands which facilitate the use of the 6502 BREAK instruction as a debugging breakpoint.
2. Hex to decimal and decimal to hex commands.
3. A Hex/ASCII memory dump command.
4. "Click on" and "click off" control of a keypress click simulator.
5. Commands that enter Applesoft, Integer, Pascal, or CP/M.
6. Commands that connect and disconnect DOS 3.3 or ProDOS.
7. A command to transfer the DOS 3.3 or ProDOS from EPROM on a firmware card to RAM.
8. A dump screen to printer command that links to your printer driver on a peripheral card I/O STROBE' ROM.

The possibilities are endless.

You can modify the system monitor without changing the command table. One idea is to change the ESCAPE handler to recognize special functions. For example, you can assign ESCAPE-G to Graphics, ESCAPE-T to Text, and so on to give yourself control of the screen modes from BASIC, the monitor, and many other keyboard polling programs.

*Address references in this application note are valid in the original Apple IIe firmware, but some are not valid in Apple's recently released Apple IIe firmware upgrade. Note particularly that, in a preliminary version of the enhanced firmware, the diagnostics reside at $C600—$C7FF, some monitor and Applesoft cassette read/write routines reside at $C500—$C5FF, no ROM check is performed in the diagnostics, and the STEP and TRACE spots in the monitor command tables are taken by SEARCH and MINI-ASSEMBLER commands. Check your listing before you attempt to apply the techniques of this application note to an enhanced firmware Apple IIe.

```
FA62: 2C 10 C0   BIT $C010   RESET KEYSTROBE; READ AKD
FA65: 10 03      BPL $FA6A
FA67: 4C 59 FF   JMP $FF59   JUMP TO MONITOR IF AKD
FA6A: D8         CLD         DO IIe RESET STUFF
FA6B: 20 84 FE   JSR $FE84   SETNORM
FA6E: 20 2F FB   JSR $FB2F   INIT
FA71: 20 93 FE   JSR $FE93   SETVID
FA74: 20 89 FE   JSR $FE89   SETKBD
FA77: A0 05      LDY #5
FA79: 20 B4 FB   JSR $FBB4   GOTOCX: INDEX 5
FA7C: AD FF CF   LDA $CFFF   I/O STROBE' ROM OFF
FA7F: C3         DFB $C3     CHECKSUM COMPENSATION
FA80: EA         NOP
FA81: EA         NOP
```

Here is an example of a change to the system monitor that does not delete any routines or capabilities but would still benefit the Apple IIe user. It modifies the RESET routine to begin by checking the AKD flag to see if a matrix key is being held down. If a key is not being held down, then the normal Autostart reset is performed. If a key is being held down, then the old monitor reset is performed. With this modification in firmware, the user may cause a reset entry to the monitor anytime he wishes by holding a convenient key while pressing then releasing CONTROL-RESET, but the normal reset is still the Autostart reset.

To create an EPROM with this modification, use the monitor MOVE command to transfer the E0—FF ROM contents to the area of RAM used as the output buffer of your PROM burner. Then, beginning at the address corresponding to $FA62, store the code above. These addresses are identical in both the original Apple IIe monitor and in the 1985 firmware upgrade.

This modified reset handler takes advantage of some redundancies in the Apple IIe monitor reset handler to make room for an AKD check and a checksum compensation byte. The redundancies are that decimal mode is cleared twice (at $FA62 and FA81) and that AN0 and AN1 are reset (LDA C058; LDA $C05A) even though this is done automatically in the IOU when RESET' falls. Additionally, space is saved since read access to $C010 both reads the AKD flag and resets the KEYSTROBE soft switch.

The checksum compensation value, $C3, is equal to the MOD $100 sum of the deleted data minus the new data. Therefore, the MOD $100 sum of the entire E0—FF ROM is the same as it was before modification. This will make the firmware diagnostic checksum procedure pass, and it will also enable some software that performs a checksum on the monitor to run with your modified monitor. It will not produce the correct compensation for checksum procedures based on exclusive-ORing or anything other than simple MOD $100 addition.

If your only intention is to make the firmware diagnostic checksum procedure pass, you can directly modify the procedure check byte location rather than using a separate compensation byte. The C1—DF ROM check byte location is $C400, and the E0—FF ROM check byte location is $F7FF.

Programs that require checksum or other verification of motherboard ROM before they will work correctly are one of the more nettlesome developments in Apple software publishing. I am sorry to say that Apple Computer, Inc. has taken up this annoying practice, and that ProDOS will not boot with many ROM modifications. The best way around the checksummers is to place your modified monitor, coresident with an unmodified Apple IIe monitor, in a 27128 EPROM instead of a 2764. Install your double monitor EPROM through a socket/adaptor which has pin 26 (A13) bent out and connected to +5 volts or ground via a manual switch. This will enable you to easily switch to the standard monitor any time you need its features. With this setup, you can make some borderline changes which you wouldn't want to do if you were stuck with the modified monitor all the time.

As a final word of advice, you should tag your modified monitor so you can recognize when it is active. It can be tagged audibly by changing location $FBE5 from $0C to $16. This noticeably lowers the pitch of the Apple's BELL so that you can verify your EPROM is active by pressing CONTROL-G or RESET. You can tag your EPROM visually at power up by changing the contents of $FB09—$FB10 to an ASCII message of your choice. For example, instead of "Apple II" you might have your screen display "KAZOO II" or some other such pertinent title. Remember to adjust your checksum compensation byte for any data that you change.

HARDWARE APPLICATION

MODIFYING A 12K FIRMWARE CARD INTO A 24K DOS HOSS

Can persons who own a 12K firmware card make use of it in the Apple IIe? Darn right. The firmware card will operate in any Apple IIe slot. If the card contains Integer BASIC and you make a minor change to DOS 3.3, you will have the ultra convenient access to Integer BASIC that was realized with a Slot 0 firmware card in the Apple II. The change to DOS consists of changing the $C08X language finding commands at $A5B7 and $A5BF to $C0nX commands where n equals the firmware card slot number plus $8.

Before you rush off to modify DOS 3.3 and install your Integer firmware card in your Apple IIe, contemplate a firmware card that gives you instantaneous access to DOS 3.3 in addition to Integer BASIC. Wouldn't it be nice to have a firmware card that contained the DOS and would alleviate the necessity of waiting for a disk boot anytime you turn on the Apple IIe. Well, if you own a 12K firmware card, a soldering iron, some 2732 EPROM, and a little time, you own that DOS/Integer firmware card. I call this super firmware card the DOS HOSS, and the purpose of this application note is to show you how to convert a 12K firmware card into a 24K DOS HOSS.

Figure 6.3 is a schematic diagram of the 12K firmware card taken from *Understanding the Apple II*. Please refer to that book for a complete description of the firmware card. Briefly, it is an INHIBIT' based, 12K firmware card with six 24-pin sockets meant to accept 2K Apple II ROMs or 2716 EPROMs. It is enabled when RESET' falls with the enable switch on, or when a program causes even access to $C0nX where n is slot number plus $8. It is disabled when RESET' falls with the enable switch off or when a program causes odd access to $C0nX.

The DOS HOSS hardware conversion consists of wiring up the unused half of the 74LS74 dual D flip-flop as a programmable bank switching flip-flop as shown in Figure 6.4.* The output of this flip-flop is tied to the A11 (pin 21) input of the EPROM sockets. This divides the card into two 12K banks, one of which is enabled every time RESET' drops (assuming the enable switch is on). When the firmware card is modified as shown in Figure 6.4, program control is via access to the following addresses with slot number times $10 in the X-register:

ADDRESS	FUNCTION
$C080,X	HOSS ON, BANK 0
$C081,X	HOSS OFF, BANK 0
$C082,X	HOSS ON, BANK 1
$C083,X	HOSS OFF, BANK 1

When the modified firmware card is installed (with the enable switch in the up position), it intercepts and processes the Apple IIe system reset. You may handle resets any way you like, but there must be a reset vector at $FFFC/$FFFD of bank 0 pointing to some sort of reset handler. The concept that is presented here is that of a DOS/Integer "hoss" that transfers DOS 3.3 and Integer BASIC to their normal locations in RAM at power up and other times selected by the operator.

Figure 6.5 is a map of the DOS HOSS. Bank 0 is filled with Integer BASIC, an Apple IIe monitor with special reset vector, and the main body of the reset handler. DOS 3.3 is in bank 1, as well as a MOVEDOS routine that transfers the DOS to RAM. The main reset handler processes all resets, and, if transfer of the DOS to RAM is required, switches control to the bank 1 MOVEDOS routine, and receives control after the DOS transfer is complete.

The DOS HOSS reset handler and MOVEDOS routines are shown in Figures 6.6 and 6.7. With this reset handler, there are three types of resets. A **special reset** occurs when the operator holds down a matrix key while CONTROL-RESET is pressed (as signaled by the AKD flag). A **power-up reset** occurs when a matrix key is not being held and the contents of power-up location ($3F4) are not equal to the exclusive-OR of $A5 and the contents of $3F3. If the power-up byte is okay and no matrix key is being held, the **normal reset** handler at $FA62 of the motherboard is executed.

The power-up reset consists of transferring DOS 3.3 and Integer to their normal RAM locations and initializing the DOS. A disk boot is not performed, so your disk drive will not automatically start up at power up when the DOS HOSS is installed. It doesn't need to because the DOS is instantly available in RAM at power up, just as if it were built into the the Apple.

The special resets are some actions which it is occasionally convenient for an operator to force. The reader is encouraged to expand or improve these special resets since there is a lot of room for more

*Please read the NOTE OF CAUTION at the beginning of the book before making any modification to your hardware.

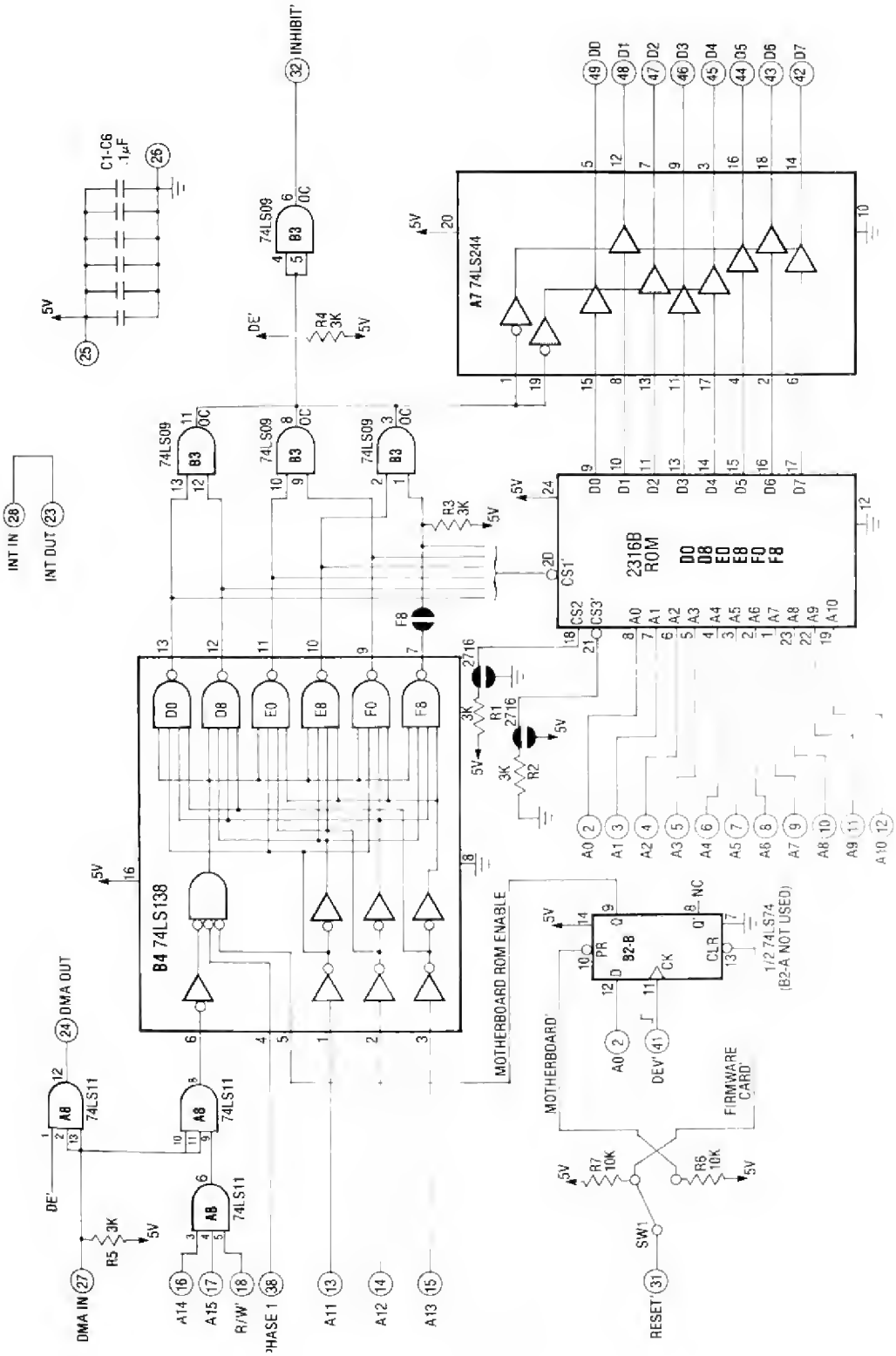


Figure 6.3 Schematic: The 12K Firmware Card.

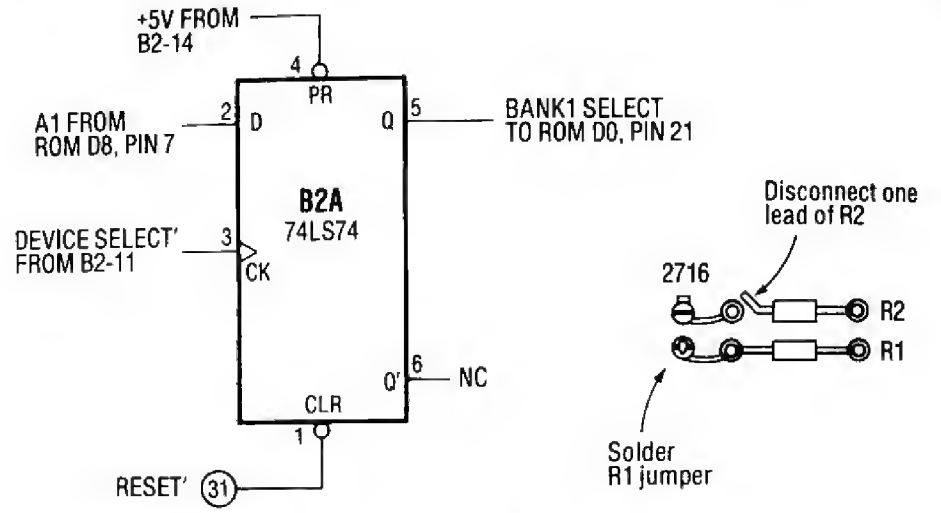


Figure 6.4 Converting a 12K Firmware Card into a 24K DOS HOSS.

code in the DOS HOSS. The special resets supported by the Figure 6.6 listing are as follows:

RESET	ACTION
OPEN APPLE	Forced disk boot
CLOSE APPLE	Diagnostic execution
C	Catalog disk
I	Integer transfer only
M	Enter monitor
B	(Boot only) transfer DOS only
H	Execute disk program named HELLO
A (or other)	Forced power-up reset

The reset handler is a simple keyboard command interpreter and executer. It begins at $D80A, so the contents of $FFFC/$FFFD in the bank 0 F8 ROM should be $0A, $D8. There are a couple of uncomplicated bank switching protocols used. First, $D000–$D809 in bank 0 and bank 1 is reserved for transferring control between bank 0 and bank 1 when the bank 1 MOVEDOS routine is called. Second, a $60 (RTS) located at $DB02 of Applesoft is used to accept program flow on the motherboard when the DOS HOSS is turned off. A motherboard routine is selected for execution by placing its address minus one on the stack before exiting to the motherboard RTS.

Program control of the DOS HOSS in Figures 6.6 and 6.7 is via fixed slot commands. You must select the slot in which the DOS HOSS resides at lines 38 and 39 of Figure 6.6 and line 19 of Figure 6.7 before assembly.

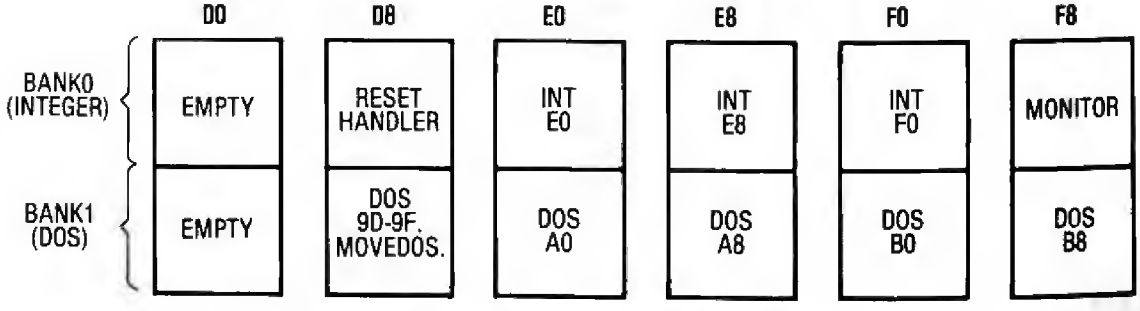


Figure 6.5 The DOS HOSS Memory Map.

```
SOURCE FILE: D8ROUTS
0000:            1 ****************************************
0000:            2 *                                      *
0000:            3 *        DOS HOSS RESET HANDLER        *
0000:            4 *                                      *
0000:            5 *             BY JIM SATHER            *
0000:            6 *                                      *
0000:            7 *             JULY 29,1983             *
0000:            8 *                                      *
0000:            9 ****************************************
0000:           10 *
0000:           11 *
0033:           12 PROMPT  EQU  $33
0036:           13 CSWL    EQU  $36
003C:           14 A1L     EQU  $3C
003D:           15 A1H     EQU  $3D
003E:           16 A2L     EQU  $3E
003F:           17 A2H     EQU  $3F
0042:           18 A4L     EQU  $42
0043:           19 A4H     EQU  $43
0045:           20 BFFRFLG EQU  $45       NO DOS BUFFERS AVAILABLE IF 0.
00D8:           21 ONERR   EQU  $D8
03CF:           22 DOSWARM EQU  $3CF      DOS WARM START VECTOR
03F4:           23 PWRUP   EQU  $3F4
A56D:           24 CATALOG EQU  $A56D
AA53:           25 DOSCSWL EQU  $AA53
AA5F:           26 DOSCMDX EQU  $AA5F
B73A:           27 DOSSTRT EQU  $B73A     DOS COLD START VECTOR.
C000:           28 KEYBRD  EQU  $C000
C010:           29 ANYKDWN EQU  $C010
C030:           30 SPEAKER EQU  $C030
C058:           31 AN0OFF  EQU  $C058
C05A:           32 AN1OFF  EQU  $C05A
C061:           33 OPENAPL EQU  $C061
C062:           34 SLDAPL  EQU  $C062
C080:           35 RDONLY  EQU  $C080
C081:           36 WRTONLY EQU  $C081
C082:           37 HRAMOFF EQU  $C082
C0D1:           38 HOSSOFF EQU  $C0D1     ASSUMES SLOT 5
C0D2:           39 BANK1   EQU  $C0D2     ASSUMES SLOT 5
CFFF:           40 CLRROM  EQU  $CFFF
DFFF:           41 BASIC   EQU  $DFFF     ENTER BASIC VECTOR.
FA61:           42 NRESET  EQU  $FA61     NORMAL RESET VECTOR
FB2F:           43 INIT    EQU  $FB2F
FBB4:           44 GOTOCX  EQU  $FBB4
FC58:           45 HOME    EQU  $FC58
FCA8:           46 WAIT    EQU  $FCA8
FE2C:           47 MOVE    EQU  $FE2C
FE84:           48 SETNORM EQU  $FE84
FF58:           49 MONITOR EQU  $FF58     (DISCONNECTS DOS)
FFFA:           50 NMIDSTN EQU  $FFFA
0000:           51 *
0000:           52 *
0000:           53 *        BANK 0 ROUTINES
0000:           54 *
0000:           55 * THESE ROUTINES CONTAIN THE BANK 0 RESET HANDLER.
0000:           56 *
----- NEXT OBJECT FILE NAME IS D8ROUTS.OBJ0
```

```
D800:           57         ORG  $D800
D800:AD D2 C0   58 MOVEDOS LDA  BANK1     PASS CONTROL TO BANK 1
D803:FF FF FF   59         DFB  $FF,$FF,$FF
D806:FF FF FF   60         DFB  $FF,$FF,$FF
D809:60         61         RTS            RECEIVE CONTROL FROM BANK 1
                62 ************************************************************
D80A:           63 *
D80A:           64 *    BEGIN RESET HANDLER
D80A:           65 *
D80A:           66 * RESET VECTOR IN F8 EPROM MUST BE $D80A.
D80A:           67 *
D80A:2C 62 C0   68         BIT  SLDAPL    START RESET HANDLER.
D80D:30 05      69         BMI  DONORM    NORMAL RESET IF SOLID APPLE.
D80F:2C 61 C0   70         BIT  OPENAPL   NORMAL RESET IF OPEN APPLE.
D812:10 0D      71         BPL  NOBUTTN
D814:A9 FA      72 DONORM  LDA  #<NRESET  GOTO $FA62
D816:48         73         PHA
D817:A9 61      74         LDA  #>NRESET
D819:48         75         PHA
D81A:4C FA DA   76         JMP  GOMRBRD
                77 ************************************************************
D81D:BD 9E      78 DOSLINK DW   $9EBD
D81F:81 9E      79         DW   $9E81
                80 ************************************************************
D821:AD F3 03   81 NOBUTTN LDA  PWRUP-1   CHECK POWER UP STATUS.
D824:49 A5      82         EOR  #$A5
D826:CD F4 03   83         CMP  PWRUP
D829:8D F4 03   84         STA  PWRUP     FIX POWER UP BYTE.
D82C:D0 05      85         BNE  DOHOSS
D82E:AD 10 C0   86         LDA  ANYKDWN   DO NORMAL RESET IF
D831:10 E1      87         BPL  DONORM    NO KEY DOWN, NOT POWER UP.
D833:           88 **
D833:D8         89 DOHOSS  CLD
D834:20 84 FE   90         JSR  SETNORM   DO A BUNCH OF RESET STUFF.
D837:20 2F FB   91         JSR  INIT
D83A:20 58 FC   92         JSR  HOME
D83D:AD 50 C0   93         LDA  AN0OFF
D840:AD 5A C0   94         LDA  AN1OFF
D843:A0 05      95         LDY  #5
D845:20 B4 FB   96         JSR  GOTOCX
D848:AD FF CF   97         LDA  CLRROM
D84B:20 22 D9   98         JSR  DUMBELL   DIFFERENT BEEP.
D84E:AD 10 C0   99         LDA  ANYKDWN   POWER UP IF NO KEY DOWN.
D851:10 3E     100         BPL  NOHELLO
D853:AD 00 C0  101         LDA  KEYBRD
D856:29 7F     102         AND  #$7F      DON'T CARE ABOUT MSB.
D858:C9 41     103         CMP  #$41      MASK LETTERS INTO CONTROL CODE.
D85A:90 02     104         BCC  *+4
D85C:29 1F     105         AND  #$1F
D85E:A8        106         TAY
D85F:C0 03     107         CPY  #$03      C?
D861:F0 44     108         BEQ  RESET.C   CATALOG
D863:A2 03     109         LDX  #3
D865:BD 53 AA  110 LINKLP1 LDA  DOSCSWL,X IF CSW AND KSW ARE EQUAL
D868:D5 36     111         CMP  CSWL,X    TO DOSCSW AND DOSKSW,
D86A:D0 0D     112         BNE  LINKDNE   ASSUME DOS WAS IN USE
D86C:CA        113         DEX            AND RECONNECT DOS.
D86D:10 F6     114         BPL  LINKLP1   THIS ONLY MATTERS IN BOOTLESS
```

Figure 6.6 Assembler Listing: 24K Firmware Card Reset/NMI Routines (1 of 2).

```
D86F:A2 03     115          LDX  #3         RESETS SUCH AS CTRL-I.
D871:BD 1D D8  116 LINKLP2  LDA  DOSLINK,X THE PURPOSE IS TO PREVENT
D874:95 36     117          STA  CSWL,X     A CRTL-I RESET FROM
D876:CA        118          DEX             DISCONNECTING DOS.
D877:10 F8     119          BPL  LINKLP2
D879:C0 09     120 LINKDNE CPY  #$09       I?
D87B:F0 4D     121          BEQ  RESET.I    INTEGER TRANSFER ONLY.
D87D:C0 0D     122          CPY  #$0D       M?
D87F:F0 59     123          BEQ  RESET.M    GO TO MONITOR.
D881:A9 1B     124          LDA  #$1B       NO HELLO FLAG.
D883:48        125          PHA
D884:C0 02     126          CPY  #02        B?
D886:F0 0F     127          BEQ  RESET.B    BOOT ONLY.
D888:68        128          PLA
D889:A9 00     129          LDA  #$0        DO HELLO FLAG.
D88B:48        130          PHA
D88C:C0 08     131          CPY  #$08       H?
D88E:F0 04     132          BEQ  DOMOVE     DO IT ALL.
D890:68        133          PLA
D891:A9 1B     134 NOHELLO LDA  #$1B       NO HELLO FLAG.
D893:48        135          PHA
D894:20 E3 D8  136 DOMOVE  JSR  MOVEINT    XFER INTEGER.
D897:20 00 D8  137 RESET.B JSR  MOVEDOS    XFER DOS.
D89A:68        138          PLA             GET DO HELLO FLAG.
D89B:8D 5F AA  139          STA  DOSCMDX
D89E:A9 B7     140          LDA  #<DOSSTRT COLD START IS $B73B.
D8A0:48        141          PHA
D8A1:A9 3A     142          LDA  #>DOSSTRT
D8A3:48        143          PHA
D8A4:4C FA DA  144          JMP  GOMRBRD
               145 ************************************************************
D8A7:A2 03     146 RESET.C LDX  #3         SET CSW AND KSW TO I/O ROUTS.
D8A9:BD 53 AA  147 CATLP   LDA  DOSCSWL,X
D8AC:95 36     148          STA  CSWL,X
D8AE:CA        149          DEX
D8AF:10 F8     150          BPL  CATLP
D8B1:A9 03     151          LDA  #<DOSWARM $3D0.
D8B3:48        152          PHA             WARM START DOS AFTER CATALOG.
D8B4:A9 CF     153          LDA  #>DOSWARM
D8B6:48        154          PHA
D8B7:A9 A5     155          LDA  #<CATALOG $A56E (DO CATALOG).
D8B9:48        156          PHA
D8BA:A9 6D     157          LDA  #>CATALOG
D8BC:48        158          PHA
D8BD:A5 D8     159          LDA  ONERR      FIX APPLESOFT ONERR FLAG
D8BF:29 7F     160          AND  #$7F       SO DISK I/O ERROR WILL PRINT.
D8C1:85 D8     161          STA  ONERR
D8C3:A9 01     162          LDA  #1         FIX NO BUFFER FLAG FOR CATALOG.
D8C5:85 45     163          STA  BFFRFLG
D8C7:4C FA DA  164          JMP  GOMRBRD
               165 ************************************************************
D8CA:A9 BE     166 RESET.1 LDA  #'>'       RESET.I = INT XFER ONLY.
D8CC:85 33     167          STA  PROMPT     FIX FOR ENTER INTEGER.
D8CE:20 E3 D8  168          JSR  MOVEINT    MOVE IT.
D8D1:A9 DF     169          LDA  #<BASIC    ENTER BASIC ($E000).
D8D3:48        170          PHA
D8D4:A9 FF     171          LDA  #>BASIC
D8D6:48        172          PHA
D8D7:4C FA DA  173          JMP  GOMRBRD
               174 ************************************************************
D8DA:A9 FF     175 RESET.M LDA  #<MONITOR GO TO MONITOR ($FF59).
D8DC:48        176          PHA             (DISCONNECTS DOS).
D8DD:A9 58     177          LDA  #>MONITOR
D8DF:48        178          PHA
D8E0:4C FA DA  179          JMP  GOMRBRD
               180 ************************************************************
D8E3:AD 81 C0  181 MOVEINT LDA  WRTONLY    ENABLE HIGH RAM FOR WRITE ONLY.
D8E6:AD 81 C0  182          LDA  WRTONLY
D8E9:A9 E0     183          LDA  #$E0       MOVE INTEGER TO HIGH RAM
D8EB:85 43     184          STA  A4H        USING THE MONITOR MOVE ROUTINE.
D8ED:85 3D     185          STA  A1H
D8EF:A9 08     186          LDA  #$00
D8F1:85 42     187          STA  A4L
D8F3:85 3C     188          STA  A1L
D8F5:A9 FF     189          LDA  #$FF
D8F7:85 3E     190          STA  A2L
D8F9:85 3F     191          STA  A2H
D8FB:A0 00     192          LDY  #0
D8FD:20 2C FE  193          JSR  MOVE
D900:A2 03     194          LDX  #3         FIX NMI AND RESET
D902:BD 1E D9  195 MOVELP3 LDA  NMISRCE,X VECTORS OF HIGH RAM
D905:9D FA FF  196          STA  NMIDSTN,X
D908:CA        197          DEX
D909:D0 F7     198          BNE  MOVELP3
D90B:A5 33     199          LDA  PROMPT
D90D:C9 BE     200          CMP  #'>'       WAS INTEGER ACTIVE?
D90F:D0 09     201          BNE  NOTINT
D911:A9 60     202          LDA  #$60       PUT A RTS IN THE RAM CARD
D913:8D 02 DB  203          STA  GOMRBRD+8 WHERE IT WILL RECEIVE FLOW.
D916:AD 80 C0  204          LDA  RDONLY
D919:68        205          RTS
D91A:AD 82 C8  206 NOTINT  LDA  HRAMOFF
D91D:68        207          RTS
D91E:FB 03     208 NMISRCE DW   $3FB       NMI VECTOR.
D920:62 FA     209          DW   $FA62      RESET VECTOR.
               210 ************************************************************
D922:A0 16     211 DUMBELL LDY  #$16       SLIGHTLY DIFFERENT BELL.
D924:A2 08     212 BELP1   LDX  #8
D926:98        213 BELP2   TYA
D927:20 A8 FC  214          JSR  WAIT
D92A:8D 30 C0  215          STA  SPEAKER
D92D:CA        216          DEX
D92E:D0 F6     217          BNE  BELP2
D930:88        218          DEY
D931:D0 F1     219          BNE  BELP1
D933:60        220          RTS
               221 ************************************************************
D934:          222 *
D934:          223 *GOMRBRD EXITS TO THE MOTHERBOARD BY TURNING THE
D934:          224 *DOS HOSS OFF.  THERE MUST BE AN RTS INSTRUCTION
D934:          225 *IN APPLESOFT AT GOMRBRD + 8 TO RECEIVE FLOW.
D934:          226 *
D934:          227          DS   $DAFA-*
DAFA:AD 10 C0  228 GOMRBRD LDA  ANYKDWN
DAFD:30 FB     229          BMI  GOMRBRD
DAFF:AD D1 C0  230          LDA  HOSSOFF
DB02:          231 *
DB02:          232 *THAT'S ALL THERE IS.
DB02:          233 *

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Figure 6.6 Assembler Listing: 24K Firmware Card Reset/NMI Routines (2 of 2).

```
SOURCE FILE: MOVEDOS
0000:            1 ****************************************
0000:            2 *                                      *
0000:            3 *              MOVE DOS                *
0000:            4 *                                      *
0000:            5 *            BY JIM SATHER             *
0000:            6 *                                      *
0000:            7 *            JULY 29,1983              *
0000:            8 *                                      *
0000:            9 ****************************************
0000:           10 *
0000:           11 *
0000:           12 *
0006:           13 SRCE     EQU  $6
0007:           14 SRCEHI   EQU  $7
0008:           15 DSTN     EQU  $8
0009:           16 DSTNHI   EQU  $9
9E25:           17 MOVEP3   EQU  $9E25
9E41:           18 RTSTEMP  EQU  $9E41
C0D0:           19 BANK0    EQU  $C0D0      ASSUMES SLOT 5
0000:           20 *
0000:           21 *
0000:           22 *          BANK 1 ROUTINES
0000:           23 *
0000:           24 * THIS ROUTINE ACCEPTS FLOW FROM BANK 0 AND MOVES
0000:           25 * DOS FROM THE DOSS HOS TO RAM.  AFTER THE
0000:           26 * MOVE, PROGRAM CONTROL IS PASSED BACK TO BANK 0.
0000:           27 *
0000:           28 *
----- NEXT OBJECT FILE NAME IS MOVEDOS.OBJ0
D800:           29          ORG  $D800
D800:FF FF FF   30          DFB  $FF,$FF,$FF
D803:20 09 D8   31          JSR  DOSAWAY    ACCEPT CONTROL FROM BANK 0
D806:AD D0 C0   32          LDA  BANK0      RETURN CONTROL TO BANK 0
D809:A9 00      33 DOSAWAY  LDA  #0          MOVE DOS.
D80B:85 06      34          STA  SRCE       FIRST SET UP INDIRECT LOCATIONS.
D80D:85 08      35          STA  DSTN       SOURCE BASE INITIALLY $DD00.
D80F:A9 9D      36          LDA  #$9D       DESTINATION BASE INITIALLY $9D00.
D811:85 09      37          STA  DSTNHI
D813:A9 DD      38          LDA  #$DD
D815:85 07      39          STA  SRCEHI
D817:A0 00      40          LDY  #0
D819:B1 06      41 MOVELP2  LDA  (SRCE),Y   MOVE IT.
D81B:91 08      42          STA  (DSTN),Y
D81D:C8         43          INY
D81E:D0 F9      44          BNE  MOVELP2
D820:E6 09      45          INC  DSTNHI
D822:E6 07      46          INC  SRCEHI
D824:D0 F3      47          BNE  MOVELP2
D826:A9 60      48          LDA  #$60       TEMPORARILY TERMINATE DOS FIX PAGE 3 ROUTINE.
D828:8D 41 9E   49          STA  RTSTEMP
D82B:20 25 9E   50          JSR  MOVEP3     FIX PAGE 3.
D82E:A9 A9      51          LDA  #$A9       RESTORE DOS ROUTINE.
D830:8D 41 9E   52          STA  RTSTEMP
D833:60         53          RTS

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Figure 6.7 Assembler Listing: Move DOS.

Creating the DOS HOSS 4K EPROM files consists of several steps. First, enter the source code of Figures 6.6 and 6.7 and assemble it into files named D8ROUTS.OBJ0 and MOVEDOS.OBJ0. Then create an uninitialized DOS 3.3 file as follows:

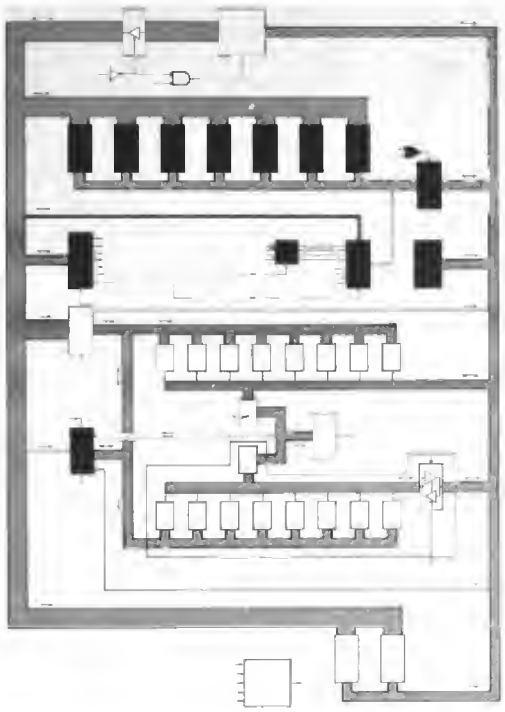
1. Boot DOS 3.3 from a disk.
2. Store 4C 59 FF (JMP $FF59) at $B73B.
3. Insert a blank disk and initialize it with this modified DOS (INIT HELLO).
4. Boot the newly initialized disk. The monitor will be entered after the DOS is loaded.
5. Move the modified DOS to $6D00—$8FFF (6D00<9D00.BFFFM).
6. Restore patched area (873B:A2 FF 9A)
7. Boot DOS 3.3 from a disk using PR#6, not open Apple RESET.
8. Save uninitialized DOS 3.3 file (BSAVE DOS,A$6D00,L$2300).

All that remains is to pack Integer, DOS 3.3, D8ROUTS.OBJ0, and MOVEDOS.OBJ0 into five 4K files for burning to 2732 EPROM. I strongly recommend that you do this with a disk EXEC file created with your assembler editor or word processor. It is very easy to make a mistake which will be difficult to track down if you do enter all these monitor commands in immediate mode. If you use an EXEC file (or separate EXEC files for each 4K file), you will have a record of what you did, and you won't have to retype all the commands if you need to make changes.

The EXEC file shown on this page will generate all five 4K EPROM source files. Before EXECing, make sure that Integer and your desired Integer associated Apple IIe monitor are resident in high RAM. D8ROUTS.OBJ0, MOVEDOS.OBJ0, and DOS must all be resident on the same disk as the EXEC file.

Some readers may recognize a similarity between the DOS HOSS and *quikLoader*, the 512-kilobyte firmware card I designed for the Southern California Research Group. That is because the DOS HOSS is the ancestor of quikLoader. After conceiving of and implementing the DOS HOSS as an application for this book, I was so happy with the result that I decided to develop it into a commercial product. The quikLoader is considerably more ambitious since it supports much larger EPROM and contains an operating system that implements cataloging, loading, and running of relatively large firmware applications in addition to instantaneous access to DOS 3.3 and Integer BASIC. Converting the 12K firmware card into the DOSS HOSS is a great way to breathe new life into an old work horse. If, however, you do not have a firmware card available, you can get the same features and more with a *quikLoader*.

```
MON I,O,C
INT
CALL -151
BLOAD DOS,A$6D00
2000:FF
2001<2000.2FFEM
BLOAD D8ROUTS.OBJ0,A$2000
BLOAD MOVEDOS.OBJ0,A$2800
2D00<6D00.6FFFM
BSAVE INT/DOS-D8,A$2000,L$1000
*
2000<E000.E7FFM
2800<7000.77FFM
BSAVE INT/DOS-E0,A$2000,L$1000
*
2000<E800.EFFFM
2800<7800.7FFFM
BSAVE INT/DOS-E8,A$2000,L$1000
*
2000<F000.F7FFM
2800<8000.87FFM
BSAVE INT/DOS-F0,A$2000,L$1000
*
2000<F800.FFFFM
27FC:A D8
2800<8800.8FFFM
BSAVE INT/DOS-F8,A$2000,L$1000
NOMON I,O,C
```



chapter 7

Input/Output in the Apple IIe

It bears repeating. MPU command of all devices in the Apple IIe computer is via signals decoded from the address bus. All persons who program the Apple become aware of this sooner or later, and all users of the Apple can save themselves problems if they understand command by addressing. The concept of data transfer between the MPU and a memory location is very easily grasped, but it must be understood that the MPU also controls parts of the computer via the address bus, often with no related transfer of data on the data bus.

For the most part, I/O in the Apple is performed under direct control of the MPU. This includes all built-in I/O features except video output, and it also includes most conventional I/O performed via peripheral cards. The MPU controls these devices by addressing them, just as if it were addressing memory. As every memory location has a specific address, every I/O device which is directly controlled by the MPU has a specific address or range of addresses.

Now there is no "Control Address Bus" command in the 6502's repertoire. The 6502 reads from or writes to the data bus on every cycle. So what does the programmer do when he wants to toggle the speaker? He utilizes a "LDA $C030" or a "CPX $C030" or a "WHO GIVES A DARN $C030" and ignores the meaningless data bus. This is why you can program the speaker with a statement like "SOUND = PEEK(−16336)". The object is not to "PEEK" into memory. The object is to get $C030 onto the address bus, commanding the speaker to toggle. Beneath the lid of the Apple, on every cycle, whether memory or a control function is being addressed, the state of the address bus is decoded to tell the rest of the Apple what the MPU is doing.

The purpose of this chapter is to discuss the various I/O devices that are built into the Apple IIe, and to show how addresses are decoded to give the MPU control over I/O processes. Related subjects such as peripheral slot capabilities and I/O firmware are discussed. Video output generation is not discussed, but is the sole topic of the following chapter.

PERIPHERAL ADDRESS DECODING CIRCUITRY

Address decoding in the Apple is the process of selecting one of 65,536 addressed locations from a 16-bit address. The overall scheme of address decoding in the Apple IIe is to divide the memory map into

the major categories of RAM, ROM, and I/O inside the MMU. Signals from the MMU are used to enable data communication and to enable further address decoding within the addressed category. These data bus management signals were described in Chapter 5. The ones which pertain to I/O are CXXX (I/O ENABLE), KBD', and MD IN/OUT'. KBD' enables the contents of the keyboard ROM to the data bus when read access is made to $C000—$C01F (Figure 7.4). MD IN/OUT' controls the direction of the peripheral slot data bus driver (Figure 7.6). CXXX is the overall I/O enable signal that enables further address decoding of I/O signals (Figure 7.1).

CXXX goes high when an address in the $CXXX range is detected on the address bus. It is not PHASE 0 gated but follows the address bus after MMU propagation delay. CXXX will not go high if INTCXROM is set and $C100—$CFFF is addressed, if SLOTC3ROM is reset and $C3XX is addressed, or if INTC8ROM is set and $C800—$CFFF is addressed.

The CXXX signal is applied to the **peripheral address decoding** circuitry where it enables further decoding of the address bus. All signals generated here are PHASE 0 gated and active when low. The circuitry is LSTTL, so an active signal will fall after PHASE 0 rises and rise after PHASE 0 falls without the long delays associated with MOS chips. Signals generated in the peripheral decoding circuitry include

(C800—CFFF)'	I/O STROBE'
C1XX'—C7XX'	I/O SELECTS'
C09X'—C0FX'	DEVICE SELECTS'
C07X'	TIMER TRIGGER'
C06X'	SERIAL INPUT ENABLE'
C04X'	C040 STROBE'
C0XX'	IOU DECODE ENABLE'

The DEVICE SELECTS', I/O SELECTS', and I/O STROBE' are connected to the peripheral slots where they can control peripheral cards. Decoding these signals on the motherboard, rather than on the cards themselves, eliminates redundant hardware on the peripheral cards. It also makes it easy to design cards so they will operate in any slot.

C07X' triggers the four paddle timers when $C07X is detected on the address bus. C06X' causes one of eight serial inputs to be placed on D7 of the peripheral data bus (Figure 7.2). C04X' is connected directly to pin 5 of the game I/O socket, and is the C040 STROBE' output of the Apple.

The C0XX' signal is connected to pin 30 of the IOU and enables address decoding in the IOU. Note the contrast here between the MMU and the IOU. The MMU monitors the entire address bus and decodes the data bus management signals which control the overall memory map. The IOU, however, monitors only parts of the $C0XX range to perform limited I/O control and display configuration functions.

The peripheral address decoding hardware is very simple, consisting of a NAND gate, a 74LS138 3 to 8 decoder, and a 74LS154 4 to 16 decoder (Figure 7.1). The NAND gate brings I/O STROBE' low during PHASE 0 when CXXX and A11 are high. I/O STROBE' thus drops low during access to $C800—$CFFF when INTCXROM and INTC8ROM are reset.

During PHASE 0 when All is low, the 3 to 8 decoder enables one of eight CnXX' signals. These are the seven I/O SELECT' lines and C0XX'. C0XX' enables further address decoding in the 4 to 16 decoder and the IOU. The 4 to 16 decoder generates the seven DEVICE SELECT' signals ($C0nX where n = $9 to $F), C04X', C06X', and C07X'. The scheme for decoding the $C0XX range is that the 4 to 16 decoder generates all the signals that simply represent a 16-byte address range. All other $C0XX signals are decoded in the IOU.

There are several interesting subtleties to the peripheral address decoding connections.

1. CXXX must be high before peripheral address decoding and IOU address decoding are enabled. It is, therefore, possible for the MMU to inhibit any of the address decoding pictured in Figure 7.1. However, the only signals which the MMU ever disables are I/O STROBE' and the I/O SELECTS' (in conjunction with INTCXROM, SLOTC3ROM, and INTC8ROM).
2. PHASE 1 is connected to an active low enable input to B5, so PHASE 0 high is a prerequisite for any address decoded action in the $C000—$C7FF range. Then why is PHASE 1 low gating also connected to C10? Connecting PHASE 1 directly to C10 results in a quicker cutoff of outputs of this chip after PHASE 0 falls. This results in operation nearly identical to that of the Apple II in that the DEVICE SELECTS' and C06X' do not linger past PHASE 0.*
3. R/W' gating is not connected to the peripheral address decoding circuitry. Therefore, command of Apple I/O features can be via read or

*Compatibility between the Apple II and Apple IIe is desirable, but I feel that the DEVICE SELECT' signals should have been allowed to linger a bit past PHASE 0 in both machines. Since the DEVICE SELECTS' are the primary data bus management gates for the peripheral slots, they should straddle the falling edge of 6502 PHASE 2.

write access. However, if you write to the $C06X serial input range, the serial input multiplexor will compete with the bidirectional peripheral data bus driver for control of D7 of the peripheral data bus. This is an undesirable situation which should be avoided.

IOU SOFT SWITCHES

The involvement of the **IOU** in the I/O processes of the Apple IIe is not extensive if you exclude video generation from consideration. Actually, the IOU is a custom video controller with a little bit of I/O control circuitry thrown in. The little bit of I/O control includes cassette and speaker toggle lines, four annunciator soft switches, the keyboard soft switch, and provisions for reading the keyboard AKD (Any Key Down) line. These functions are grouped together with the display mode soft switches in Figure 7.1. This figure depicts all soft switches and address decoding functions of the IOU.

Table 7.1 shows the addresses which control the IOU soft switches. Most of the soft switches have a set address and a reset address. Exceptions are the **SPKR** line which is toggled by access to $C03X, the **CSSTOUT** line which is toggled by access to $C02X, and the **KEYSTROBE** soft switch which is set by a keypress or the auto repeat strobe and reset by write access to $C01X or any access to $C010. The **AKD** line and **VBL'** line are not programmable soft switches, but they can be read similarly to the soft switches. AKD and VBL' are gated to MD7 of the data bus by read access to $C010 and $C019 respectively for reading by the MPU.

The 80STORE, 80COL, ALTCHRSET, TEXT, MIXED, PAGE2, and HIRES soft switches control the display mode of the Apple IIe. They accomplish this function by controlling which memory is scanned for video output (Figure 5.3), by controlling timing generation (Figure 3.9), and by controlling address inputs to the video ROM (Figure 8.5). The display mode soft switches are discussed in greater detail in Chapter 8.

The annunciators, CSSTOUT, and SPKR lines are output from the IOU. The annunciators are connected directly to the game I/O socket, and CSSTOUT and SPKR are processed then applied to the cassette output jack and speaker.

The KEYSTROBE soft switch performs no function other than to indicate to the MPU program that a key has been pressed or that rapid keypresses are being simulated by the auto repeat circuitry (Figure 3.8). In other words, the KEYSTROBE soft switch exists only to be read by the MPU, not to control functions internal to the IOU. Likewise, the AKD line is routed through the IOU so it can be read by the MPU.

When read access is made to $C000—$C01F, KBD' from the MMU goes low and the ASCII of the latest keypress is output from the keyboard ROM to MD0—6 of the data bus. This word is filled out, on MD7 of the data bus, by KEYSTROBE ($C00X), AKD ($C010), or one of the soft switches read at $C011—$C01F. Thus, as far as the controlling program is concerned, read access to $C000—$C01F fetches an 8-bit word with keyboard ASCII in the lower 7 bits and the state of an IOU or MMU flag in the most significant bit.

As with all drawings in *Understanding the Apple IIe* that show circuits internal to the MMU and IOU, the IOU circuits of Figure 7.1 are only a functional representation resulting primarily from my investigations of Apple IIe operational features. Hopefully Figure 7.1 is functionally correct, but it does not correctly depict such circuit details as true arrangement of logic gates and flip-flops. Explanations of some of the functional details of the IOU portion of Figure 7.1 follow here.

1. All soft switches except KEYSTROBE, TEXT, and MIXED are reset when the RESET' line drops low. All soft switches appear to be cleared at power up.
2. AL0—AL5 and AL7 are address bus values A0—A5 and A7, latched from the RAM address bus at RAS' falling during PHASE 0 (Figure 5.3).
3. There appears to be a window during which C0XX', A6, R/W', AL0—AL5, and AL7 are monitored for commands which determine soft switch states. If a soft switch command is held valid at the IOU inputs for about 40 nanoseconds or more during the window, the affected soft switch will respond to the command. As well as I can determine, the IOU soft switch window is PHASE 0 • RAS'' • Q3. The 6502 address valid period and consequent C0XX' valid period completely overlap this window.
4. AKD is delayed by two MPU cycle periods before application to MD7. KSTRB is delayed by two MPU cycle periods and quantized to one MPU cycle period before application to the KEYSTROBE set input. It is my belief that the delay serves no function since keyboard ASCII is readable approximately 10 milliseconds after AKD goes high and 20 microseconds before the KSTRB pulse. Quantizing KSTRB to one MPU period prevents KEYSTROBE from setting

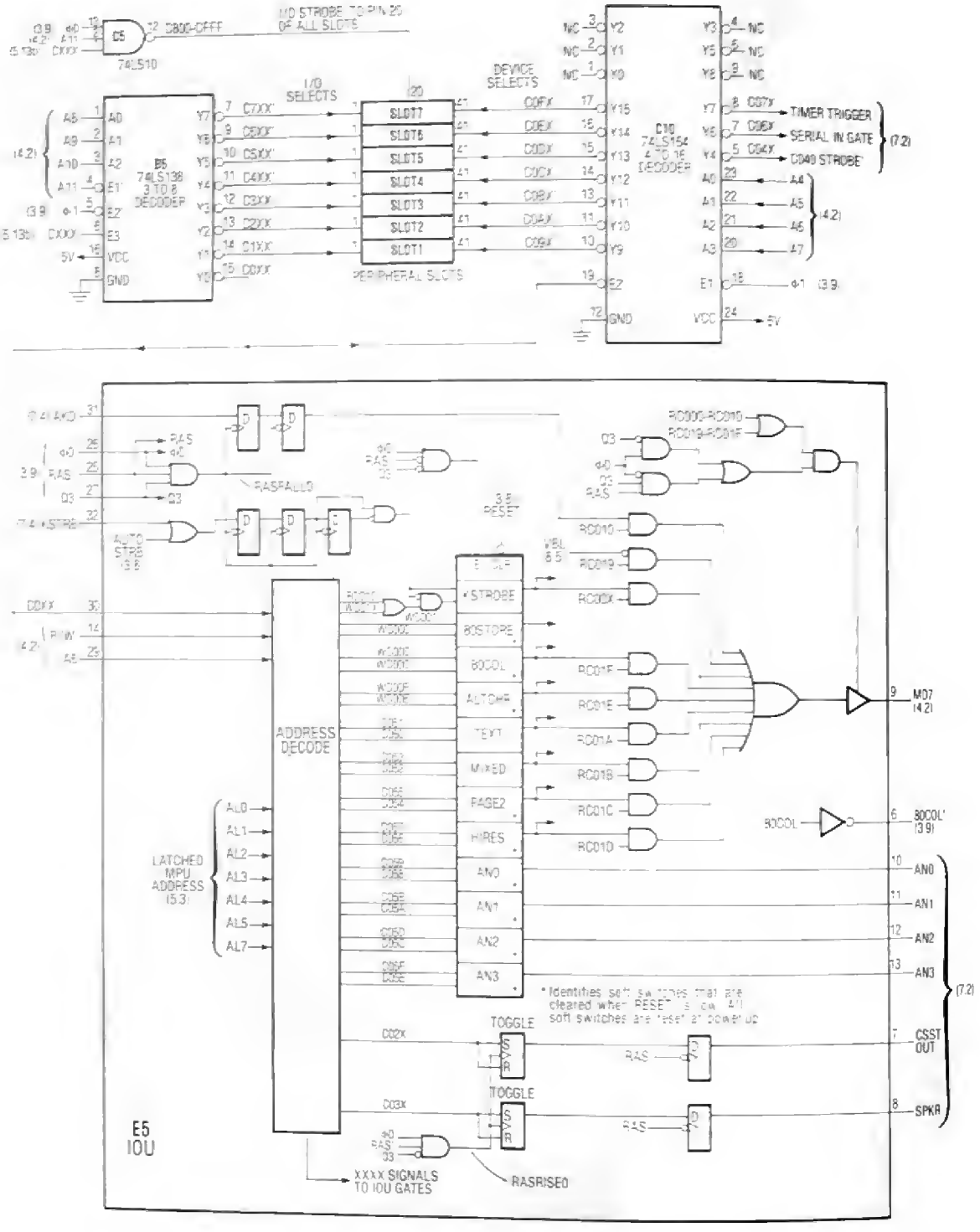


Figure 7.1 Schematic: IOU Soft Switches and Peripheral Address Decoding Circuitry.

Table 7.1 Address Control of IOU Soft Switches.

SOFT SWITCH	OFF ADDRESS	ON ADDRESS	READ ADDRESS	CONDITION AFTER RESET'
KEYSTROBE	$C010/ W$C01X	KSTRB/ AUTOSTRB	R$C00X	NA (NOT APPLICABLE)
80STORE*	W$C000	W$C001	R$C018	PAGE2 SWITCHES DISPLAY AREA
80COL	W$C00C	W$C00D	R$C01F	SINGLE-RES DISPLAY
ALTCHRSET	W$C00E	W$C00F	R$C01E	FLASHING TEXT ACTIVE
TEXT	$C050	$C051	R$C01A	NA
MIXED	$C052	$C053	R$C01B	NA
PAGE2*	$C054	$C055	R$C01C	PAGE1
HIRES*	$C056	$C057	R$C01D	LORES
AN0	$C058	$C059	NA	OFF
AN1	$C05A	$C05B	NA	OFF
AN2	$C05C	$C05D	NA	OFF
AN3	$C05E	$C05F	NA	OFF
CSSTOUT**	$C02X	$C02X	NA	NA
SPKR**	$C03X	$C03X	NA	NA
AKD	NA	NA	R$C010	NA
VBL'	NA	NA	R$C019	NA

NOTES:
* PAGE2, HIRES, and 80STORE are mechanized identically in the MMU and IOU. The MMU passes the state of 80STORE to MD7 when $C018 is read, and the IOU passes the state of PAGE2 or HIRES to MD7 when $C01C or $C01D is read.
** CSSTOUT and SPKR are toggled when their control addresses are accessed.

more than once from one KSTRB pulse. My investigations indicate that the delay function is clocked by RAS' falling during PHASE 0.

5. CSSTOUT or SPKR line toggling appears to occur if C02X or C03X is valid in the IOU at RAS' rising during PHASE 0. The line does not actually toggle, however, until after RAS' falls during the following PHASE 1.
6. When the MPU reads the state of an IOU soft switch, the MD7 enable gate appears to be PHASE 0 • Q3' + PHASE 0' • Q3 • RAS', which is the last three 14M periods of PHASE 0 and the first 14M period of the following PHASE 1.

SERIAL I/O HARDWARE

Figure 7.2 is a schematic of Apple's serial I/O devices. Most of these devices are connected to the outside world through the **game I/O socket** (J15) and the **game I/O extension connector** (J8). The pins on the game I/O socket and extension connector are available to external devices such as joysticks with pushbuttons. Additionally, the four annunciator signals and the C040 STROBE' are tied directly to pins of the game I/O socket.

Apple IIe serial inputs are connected to D7 of the peripheral data bus through the 74LS251 **serial input multiplexor**. The LS251 is an 8 to 1 multiplexor whose tri-state output is enabled when C06X' from the peripheral address decoding circuitry drops low. A0, A1, and A2 from the address bus are addressing inputs to the multiplexor, so access to $C06X results in one of eight addressable inputs being passed to D7. When read access is made to $C06X, the MMU brings MD IN/OUT' high and the addressed serial input is passed to MD7 for reading by the MPU. Each input can be read at two addresses, but programming convention is to read the serial inputs only at the low addresses ($C060—$C067). The eight serial inputs and their addresses are:

Cassette input	$C060/$C068
Pushbutton 0	$C061/$C069
Pushbutton 1	$C062/$C06A
Pushbutton 2	$C063/$C06B
Paddle 0	$C064/$C06C
Paddle 1	$C065/$C06D
Paddle 2	$C066/$C06E
Paddle 3	$C067/$C06F

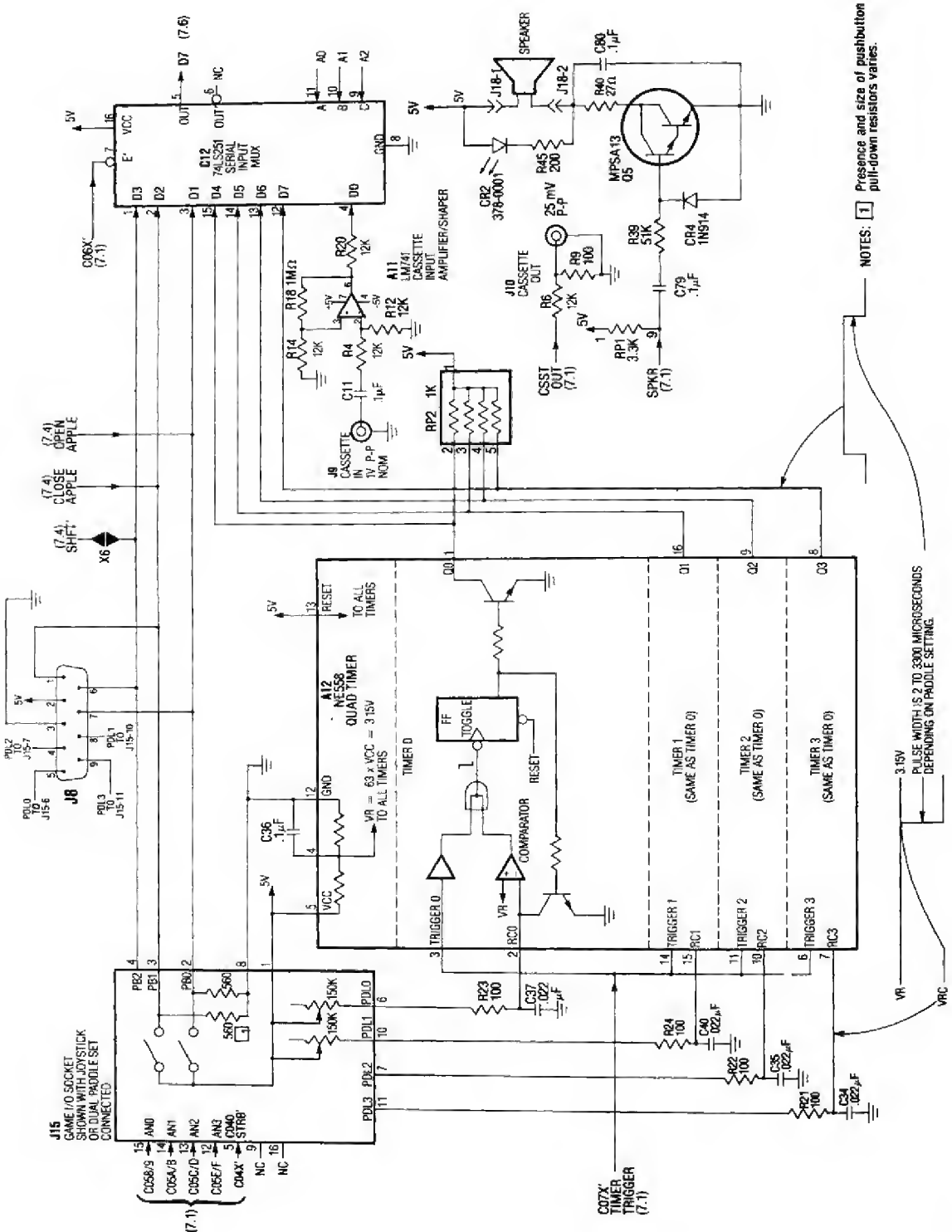


Figure 7.2 Schematic: Serial I/O Devices.

The serial input multiplexor is a good example of how things come in groups of eight or other powers of two in digital computers. Three address lines can be in eight different states, so 8 to 1 multiplexors exist for computer designers to use. This naturally leads to a computer with eight serial inputs. There is an operational oddity in the Apple that there are four paddle inputs but only three pushbutton inputs. It becomes logical in the light of hardware convenience. Eight serial inputs minus four timer inputs minus one cassette input leaves room for three pushbutton inputs.

A standard paddle set or joystick is shown connected to the game I/O socket in Figure 7.2. This illustrates the fact that plugging a standard paddle set into the game I/O socket makes eight serial input or output ports inaccessible there. If the paddle set were shown attached to the extension connector, the same four I/O lines would be utilized. There seems to be an unnwritten law in the Apple world that only PDL0, PDL1, PB0, and PB1 will be commercially supported. There are exceptions, but that is the general rule.

The three **pushbutton** inputs (PB0, PB1, and PB2) are actually TTL inputs to the serial input multiplexor. They could be used for all sorts of serial input but are normally used for pushbuttons. PB0 is also connected to the **open Apple** switch on the keyboard, and PB1 is also connected to the **close Apple** switch. Both the PB0 and PB1 lines are pulled down to ground by 470 ohm resistors on the keyboard (Figure 7.4). When open Apple or close Apple is pressed, 5 volts is applied to the LS251 input. When the key is released, the LS251 input is pulled below the LSTTL low voltage threshold by the 470 ohm resistor.

In addition to the pull-down resistors mounted on the keyboard, most paddle sets and joysticks will have PB0 and PB1 pull-down resistors mounted in them. This is a throwback to the Apple II which did not have pull-down resistors installed as the Apple IIe does. The value of the pull-down resistors varies depending on manufacturer but will be found to be in the 200—1000 ohm range. Since this resistance is in parallel with 470 ohm resistors on the keyboard, the resistive load on the 5 volt line is 140—320 ohms when a pushbutton or open/close Apple key is pressed. If a paddle and joystick with 200 ohm pull-down resistors are installed in a game I/O extender, and PB0 and PB1 are pressed simultaneously, the 5 volt line is connected to ground through a resistance of 45 ohms. This will cause the current load on the 5 volt line to increase by over 100 milliamps while the two buttons are pressed. This should cause no problems in most Apples, but it is something to think about if your heavily loaded computer crashes sometime when you press a pushbutton.

The Apple II **shift key mod** is available via jumper pad X6 on the Apple IIe motherboard. The shift key mod evolved as a method for distinguishing between upper and lower case keyboard input on the upper case only Apple II. The modification consists of connecting the keyboard SHIFT line to PB2. A program can then tell when a shift key is being held down and thus distinguish between upper and lower case keyboard entry. If you possess some software which recognizes the shift key mod, but not upper/lower case ASCII from the keyboard port, you can utilize your software in the Apple IIe by soldering the X6 pad. If you are not currently using the PB2 input at all, you may wish to solder X6 so you can use PB2 for your own purposes. With X6 soldered, you can activate all three pushbutton inputs from the keyboard.

The four **paddle** inputs (PDL0, PDL1, PDL2, and PDL3) from the game I/O socket and extension connector are tied to a **quad timer**. These inputs are not high/low binary voltage inputs like the pushbuttons. For that matter, the timers are not digital devices, although they do have TTL compatible outputs to the serial input multiplexor.

The way each timer works is this: first, the timers are triggered by $C07X access. The outputs of the four timers then go high and each one stays high for a period of time determined by the position of the paddle connected to its input pin. The paddle knobs are attached to 150,000 ohm potentiometers (variable resistors). Each of these potentiometers is part of an R/C (Resistance/Capacitance) network which determines the pulse width of one of the outputs of a quad timer. They are designed so that the pulse width in microseconds will be equal to (RP + 100) x .022 where RP is the resistance of a paddle. Since the paddles are 150,000 ohm pots, the timer outputs can be varied from 2 (100 x .022) to 3300 (150100 x .022) microseconds.

The four timer outputs are connected to the serial input multiplexor so their high/low states can be read by a program. The programming method is to trigger the four timers, then poll the output of the pertinent timer in a loop while counting the program loops before the timer resets. Of course, a standard paddle set uses only two of the four available timers.

The NE558/SE558 quad timer has a RESET input which brings all four outputs low and inhibits

triggering. This feature is not utilized in the Apple. Each timer has its own input trigger, but in the Apple, all four triggers are tied to the C07X' line. To achieve the reset and individual trigger capabilities would have required extra address decoding circuitry and possibly would have decreased the performance/cost ratio of the Apple. As it is, the common trigger gives the capability of simultaneously reading two or more paddles, and the lack of reset merely requires waiting for timer reset at the beginning of timer polling routines. Unfortunately, neither of these realities is supported by Apple firmware.

The timer set up in the Apple is an astonishingly cheap way to achieve a 4-channel analog to digital input capability. It is possible to obtain variable resistors whose resistance is proportional to light, heat, linear motion, rotational motion, chemical composition, and probably a lot of other pertinent things. This means that you can monitor all these qualities with an Apple computer via the quad timer. Of course, it takes a while to read these resistances, 22 microseconds per thousand ohms. One the other hand, how much is the temperature going to change in a thousand microscconds?

The one thing the timers are the least suited to do is the thing they are used quite often for, game controllers for real time arcade type games. A sophisticated HIRES arcade game on the Apple requires all of a programmer's skill if the result is fast paced realistic action. The 6502 in the Apple executes an instruction every three or four microseconds. It takes about 500 times as long to perform a typical timer polling routine. Needless to say, program speed is inversely proportional to the time spent polling the timers. Were the Apple IIe designed today, without the constraint of compatibility with the older Apple II, one would suspect that a modern multichannel quick response analog to digital converter would be used for paddle input rather than timers.

The eighth input to the serial input multiplexor is the **cassette input**. The cassette input jack of the Apple is connected to a high gain amplifier/shaper. The amplifier takes the small signal from the earphone jack of a cassette player and converts it to high and low voltages which can be read correctly by the serial input multiplexor. Of course, you can still adjust the cassette player volume too high or too low, but the amplifier gives you a very reasonable chance of selecting a volume which works.

The cassette input amplifier/shaper is an electronic circuit that answers the question, "when is an operational amplifier not an operational amplifier?" The answer is that an LM741 operational amplifier is not an operational amplifier when there is no degenerative feedback. It then becomes a saturated amplifier and, in the case of the Apple cassette input, a threshold detector/signal shaper. Mild apologies for all that electronic language, but this is an electronic circuit. What it does is this:

1. Blocking capacitor C10 removes any DC component of the input voltage and makes the pin 2 input to the LM741 vary above and below 0 volts.
2. The LM741 acts like a threshold detector. If the voltage at pin 2 rises above .15 volt (very approximately), the voltage at pin 6 will go as negative as an LM741 can bring it (about −4.3V) operating from a −5 volt negative supply. If the voltage at pin 2 lowers below −.15 volt (very approximately), the voltage at pin 6 will go as positive as an LM741 can bring it (about +4.3V) operating from a +5 volt positive supply. Thus, as long as the cassette input exceeds the input threshold, the voltage at pin 6 will be an 8.6 volt p-p squared signal that switches high or low as the input crosses the low threshold or high threshold.
3. Pin 6 of the LM741 is connected to the serial input multiplexor through a 12 thousand ohm resistor. While A11-6 is at +4.3 volts, 4.1 volts or so is felt at C12-4. While A11-6 is at −4.3 volts, the negative input clamp of the LS251 holds the voltage at pin 4 very close to zero volts, and the 12 thousand ohm resistor limits input current to about .36 milliamps.

The neat thing about the cassette input circuit is that even when your cheap tape recorder distorts a digital square wave, a square wave of correct pulse width is presented to the input of the serial input multiplexor. Figure 7.3 shows what happens when a $30 tape player makes a sine wave out of your square wave. The LM741 still switches at points separated in time by multiples of the period of the program loop that stored the information to cassette. Of course, there are limits to the distortion which an Apple can work with.

The *Apple II Reference Manual for IIe Only* states that the nominal voltage required at the cassette input is 1 volt peak to peak. This is a voltage one would reasonably expect to find at the earphone output of a cassette recorder. The **cassette output** of the Apple is a much smaller voltage, comparable in amplitude to the signal out of a microphone. This voltage is the CSSTOUT signal from pin 7 of the IOU reduced by a factor of 121. The CSSTOUT signal level toggles once every access to $C02X. If

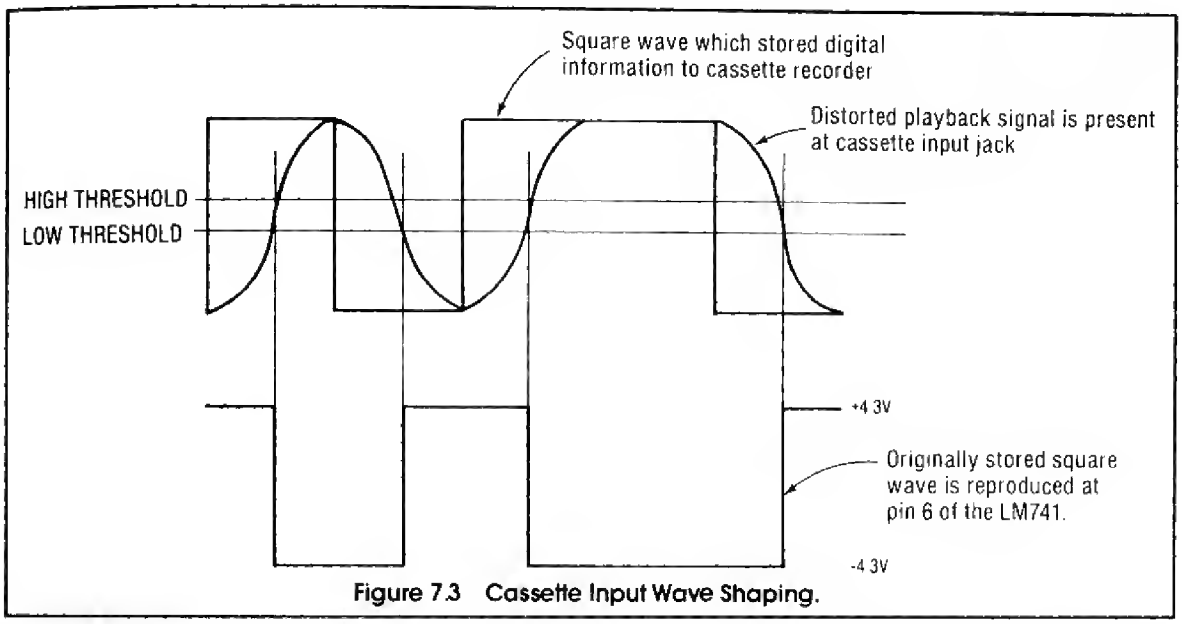


Figure 7.3 Cassette Input Wave Shaping.

you assume that CSSTOUT swings back and forth between 0 and 3 volts while it is being toggled, then the cassette output jack swings between 0 and .025 volts (3/121 = .025).

The SPKR signal from pin 8 of the IOU is a toggle signal like CSSTOUT, except SPKR toggles when $C03X is accessed. SPKR is applied to a simple audio amplifier which drives the Apple's speaker. The amplifier is necessary because the IOU cannot directly drive a 2 ¼ inch speaker. Current flow through the speaker is in only one direction so the speaker action is tense/relax rather than push/pull. Alternate references to $C03X tense the speaker diaphragm and then relax it, but the program cannot determine whether a $C03X reference causes tension or relaxation. This dampens the possibilities of complex program control of the speaker tension. In any case, an audio cycle consists of a tension half cycle and a relaxation half cycle, so two $C03X references are required per audio cycle. For example, to program a 1000 Hz tone, you reference $C03X 2000 times a second.

There is an LED (Light Emitting Diode) connected across the speaker jack on the Apple IIe motherboard. However, the LED (CR2) does not normally light because there is not enough voltage developed across the speaker primary to cause visible light emission. The light does glow when SPKR is toggling with the speaker disconnected. You can see this for yourself by disconnecting the speaker plug and holding open Apple and close Apple and pressing CONTROL-RESET. This causes firmware diagnostic execution with speaker tones or light emission from the LED when the speaker is disconnected. The LED thus provides a means of verifying diagnostic performance on a motherboard with no speaker connected.

APPLE IIe KEYBOARD CIRCUITRY

Most readers are aware that the Apple IIe keyboard is a considerable improvement over the old upper case only keyboard of the Apple II. This is due more to the deficiencies of the Apple II keyboard than any exciting features of the Apple IIe keyboard. Actually, the Apple IIe has an adequate keyboard that can input all characters needed for normal text handling applications. It lacks a built-in numeric keypad and user-definable function keys, but Apple owners seem to be able to manage without them.

Though the keyboard itself is merely adequate, there are some keyboard capabilities hidden in the motherboard circuitry. What I'm speaking of is the fact that the keyboard ASCII is mapped in a standard 2K ROM on the motherboard, and that anyone who can program a 2K EPROM can define any of the matrix keys in the Apple IIe to represent whatever ASCII he desires. Also, there is an alternate keyboard set which can be selected if you install a switch in the right place. With the keyboard ROM which comes with the American Apple, the alternate keyboard layout is the Dvorak (American Simplified) keyboard. But if you burn your own

keyboard EPROM, you can create any alternate layout you desire.

Figure 7.4 is a schematic diagram of the Apple IIe keyboard circuitry. The source of detail for this drawing is the Apple IIe schematic in the *Apple II Reference Manual for IIe Only*, except for the keyboard itself. I couldn't find a schematic of the keyboard but drew Figure 7.4 based on my investigations of a real keyboard.

There is very little electronic circuitry on the keyboard itself, just a bunch of spring loaded, normally open switches. The majority of the switches are connected in an **X,Y matrix** with eight X lines and ten Y lines connected through the motherboard keyboard connector to a special purpose IC called a **keyboard encoder**. The keyboard encoder scans through its X drivers while monitoring its Y receivers to see if a key is pressed. For example, if the right arrow key (key 61) is held down, the encoder will detect this fact because it senses Y9 active when it activates X0. When a matrix keypress is sensed, the encoder produces code that is translated by the **keyboard ROM** to ASCII.

Of the 63 keys on the Apple IIe keyboard, 56 are connected in the X,Y matrix. Keys not in the matrix perform special functions which will be described shortly. As far as the keyboard is concerned, pressing left or right SHIFT or CONTROL pulls the **SHIFT'** or **CTRL'** line low respectively. The CAPS LOCK key is similar except that it has a locking mechanism that alternately locks it closed or releases it to the open position. When CAPS LOCK is locked in the down position, the **CAPLOCK'** line is held low and the keyboard ROM produces upper case alphabetic code. Pressing open Apple or close Apple brings PB0 or PB1 high respectively. SHFT', CTRL', and CAPLOCK' are pulled up by 1K ohm resistors on the motherboard. PB0 and PB1 are pulled down by 470 ohm resistors on the keyboard.

The **RESET'** line drops low if RESET and CONTROL are pressed simultaneously. This can be changed so that RESET' drops low if only RESET is pressed by reconfiguring the two jumpers on the keyboard. These jumpers can be seen if you remove the keyboard and look at the bottom of the keyboard PC board. **Solder the normally open jumper and cut the normally closed jumper to make CONTROL not required for RESET' to fall.** I recommend doing this since the RESET key on the Apple IIe is recessed and separated from the other keys, so accidentally resetting the computer is unlikely.

SHFT' and CTRL' are connected to control inputs to the keyboard encoder so that they affect the code generated by matrix keypresses. CAPLOCK' is an input to the keyboard ROM and thus divides the ROM into a caps locked section and a caps not-locked section. RESET' is distributed to the MPU, the IOU, and the peripheral slots. While the keyboard provides a convenient physical location for the RESET key, the other keyboard circuitry is not affected by RESET'. PB0 and PB1 are inputs to the serial input multiplexor and are read at $C061 and $C062 (Figure 7.2). The open Apple and close Apple keys have no association with the other keyboard circuitry.

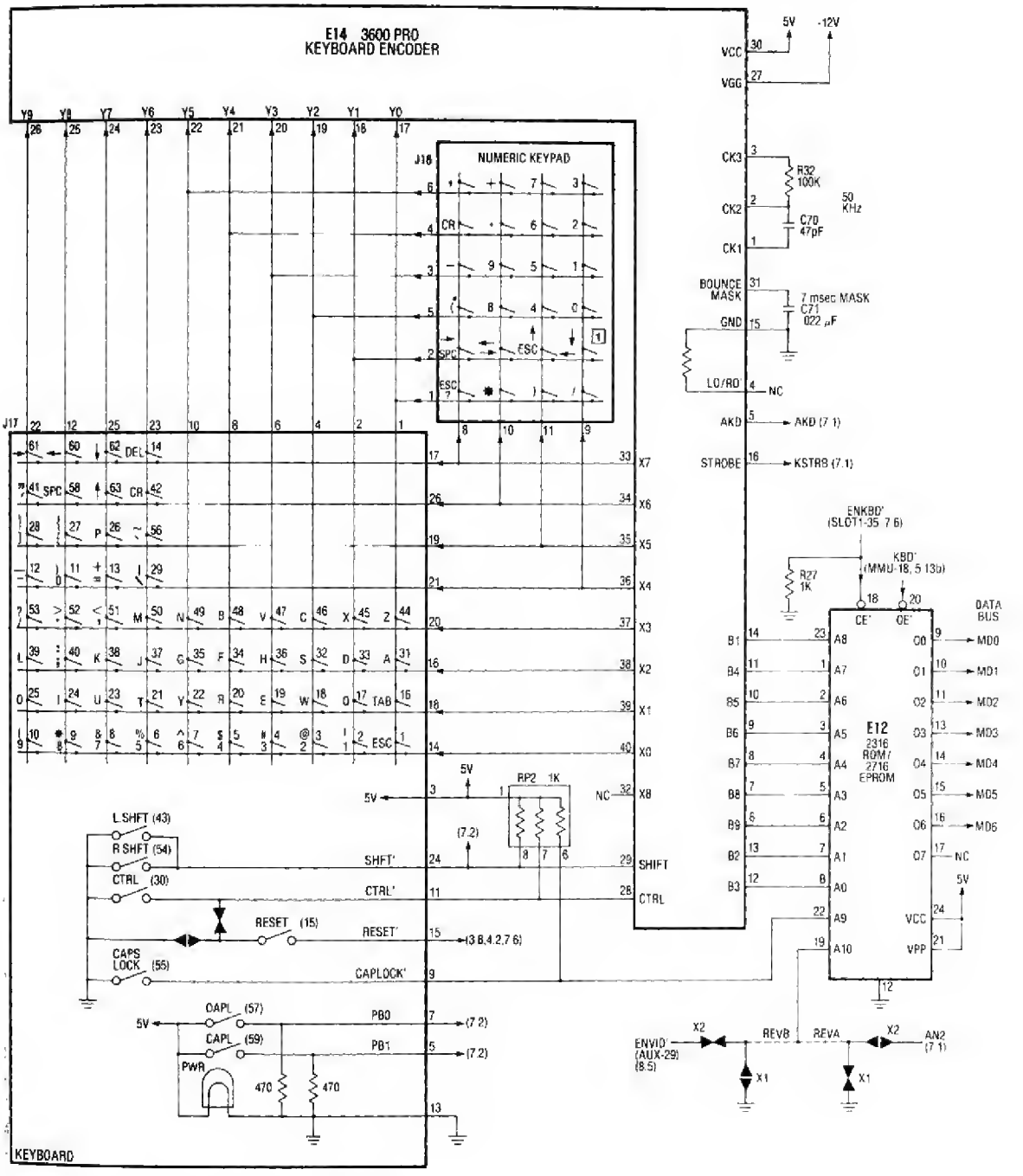
There are no matrix keys on the keyboard at the junctions of Y0—Y5 with X4—X7. These junctions are reserved for a numeric keypad which can be connected to J16 on the motherboard. The numeric keypad could share matrix junctions with the keyboard, but the Apple implementation is superior. With separate matrix junctions, the numeric keypad layout can be defined without regard to the keyboard layout.

A **numeric keypad** is shown connected to J16 in Figure 7.4. The characters shown next to the keypad junctions are those of the keyboard ROMs supplied with the Apple IIe. There are 24 junctions available for the keypad (4 x 6), and these can be completely redefined by installing a custom keyboard EPROM in place of the ROM at E12.

The Apple IIe keyboard encoder is a 3600-PRO which is a special purpose version of the general purpose 3600 encoder. The general purpose encoder is like a masked ROM in that the purchaser specifies which code is output for each junction. Also, certain options can be selected by the purchaser for pins 1—5. These options include internal or external clock oscillator, lockout/rollover' control, output complement control, chip enable control, any key down output, and extra output data bit (B10).

The 3600-PRO is a fixed version of the 3600. Options and output code are standardized, and the designer chooses his own code by running the output from the encoder through a translator ROM or EPROM. Fixed features of the 3600-PRO include an internal oscillator controlled at pins 1, 2, and 3, a lockout/rollover' select at pin 4 with internal pull down resistor, uncomplemented outputs, and AKD (any key down) output at pin 5. Other features of the 3600-PRO are fixed features of the general purpose 3600.

The keyboard encoder has an internal clock oscillator whose frequency is set by external components in the Apple IIe at approximately 50 KHz. The switch bounce mask period is also set by an external component. The bounce mask period is a delay from keypress to data and strobe output in the encoder



NOTES: [1] SPC, right, ESC, left, ? of 341(2)-0132-B keyboard ROM (early Apple IIe) were replaced by right, left, up, down, ESC in 341(2)-0132-C keyboard ROM (later Apple IIe).

Figure 7.4 Schematic: Apple IIe Keyboard Circuitry.

which prevents interpreting switch bounce as multiple keypresses (see Figure 7.5). According to the nomograph in the data sheet, a C71 value of .022 microfarads should result in a 7-millisecond mask period in the Apple IIe.* However, I measure the mask period in my Apple IIe at 13 milliseconds, so I compromise and say that switch bounce in the Apple IIe is masked for approximately 10 milliseconds.

Figure 7.5 is a timing diagram from the General Instrument AY-5-3600 data sheet. As the diagram shows, the data ready strobe is output only after a key has stopped bouncing and the mask period has passed. The strobe is a 20-microsecond (one clockpulse) positive pulse which is output once for every keypress. This strobe is the **KSTRB** input to the IOU which sets the KEYSTROBE soft switch (Figure 7.1) and resets the auto repeat delay function (Figure 3.8).

It can be seen in Figure 7.5 that the data output of the keyboard encoder is set up sometime in the clock period preceding the data ready strobe. Actually, Figure 7.5 is extremely misleading, because the data output is set up very quickly after the beginning of the clockpulse period preceding the data ready strobe. In the Apple IIe, the encoder data is actually set up for approximately 20 microseconds before the data ready strobe goes high. This is far more than the 450-nanosecond address to data propagation delay of the keyboard ROM. Knowing this, it seems strange that Apple took pains to delay KSTRB a couple of MPU cycles in the IOU. Perhaps they were misled by the 3600 data sheet timing diagram into thinking the delay is necessary so that ASCII from the keyboard ROM will be certain to precede KSTRB.

The **AKD** output of the encoder goes high when any matrix key is pressed and stays high until all matrix keys are released. This AKD output is routed to the IOU where it enables the AUTOSTRB circuitry (Figure 3.8) and is passed to MD7 when $C010 is read (Figure 7.1). Note that the AKD line goes high immediately without waiting for the mask period to pass. This means that there is a period of approximately 10 milliseconds after a key is pressed when a program will sense AKD active when reading $C010, but when the keyboard ASCII will not yet be updated to that of the current keypress. For this reason, programs should normally wait until KEYSTROBE is set before interpreting the keyboard ASCII. Using AKD for this purpose could easily lead to errors.

The **lockout/rollover'** pin of the encoder in the Apple IIe is open. Since the line is pulled low internally, Apple IIe keypresses "roll over." This means

*General Instrument publication *ROM 3*, AY-5-3600 Keyboard Encoder.

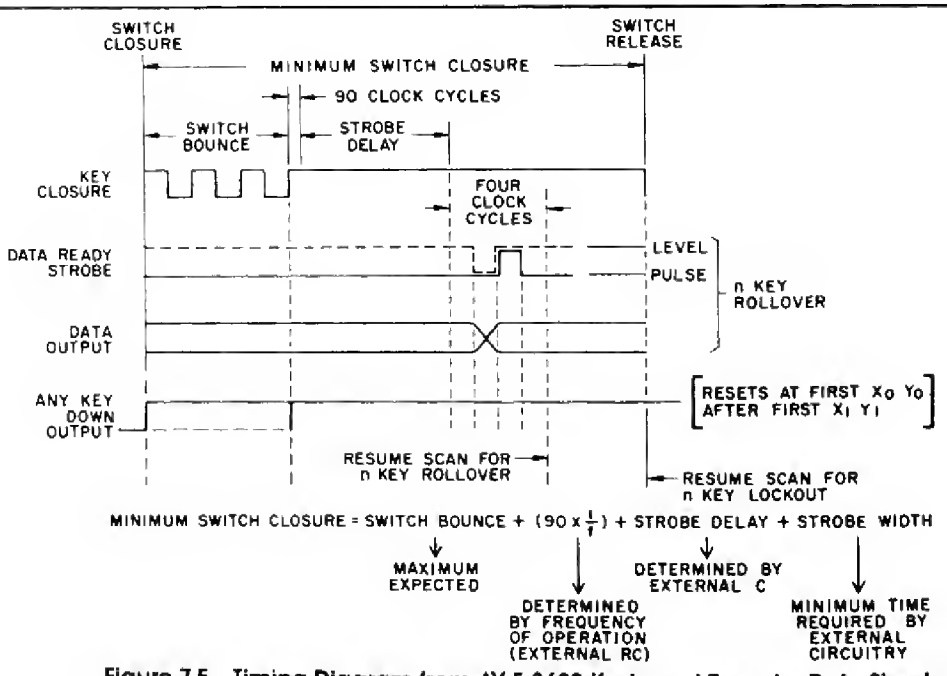


Figure 7.5 Timing Diagram from AY-5-3600 Keyboard Encoder Data Sheet (Courtesy of General Instrument Corp).

that when a key is pressed while another key is held down, the code for the new key is placed on the encoder output line after the mask period. With lockout, code from new keypresses are locked out until the old key is released. If you would like lockout rather than rollover on your Apple IIe, just jumper pin 4 of your encoder to +5 volts.

The following quote from the AY-5-3600 data sheet describes n-key rollover operation. "When a match occurs, and the key has not been encoded, the switch bounce delay network is enabled. If the key is still depressed at the end of the selected delay time, the code for the depressed key is transferred to the output data buffer, the data ready signal appears, a one is stored in the encoded key memory, and the scan sequence is resumed. If a match occurs at another key location, the sequence is repeated, thus encoding the next key. If the match occurs for an already encoded key, the match is not recognized. The code of the last key encoded remains in the output data buffer."

The output of the keyboard encoder is **latched information which cannot be modified by the MPU**. When a key is pressed and the mask period passes, the code for that matrix junction is output from the 3600-PRO. This code addresses the keyboard ROM, and the ASCII can be read at the output of the keyboard ROM no more than 450 nanoseconds after the code from the encoder becomes valid (assuming 450-nanosecond ROM or EPROM). This ASCII can be read at any time at $C000—$C01F before another matrix key is pressed and the new code is valid at the output of the encoder long enough to propagate through the keyboard ROM.

A **Slot 1** peripheral card can disable keyboard ROM response to $C000—$C01F by pulling **ENKBD'** high. This deactivates the chip enable input to the keyboard ROM. It is therefore possible for an alternate keyboard interface card in Slot 1 to steal response to $C000—$C01F. However, most alternate keyboards designed for use with the Apple IIe do not plug into Slot 1 but plug directly into the motherboard keyboard connector. The capability of disabling the keyboard ROM from Slot 1 is probably meant to support some motherboard production checkout test apparatus.

The output of the 3600-PRO forms the word B1B4B5B6B7B8B9,B2B3, where B2B3 is the shift/control **mode ident** and B1B4B5B6B7B8B9 is the **junction code**. After the mask period when a matrix key is pressed, B2B3 is latched to 11 normally, 10 if CONTROL only is also pressed, 01 if SHIFT only is also pressed, and 00 if both CONTROL and SHIFT are also pressed. This normal/control/shift/both operation is the inverse of what the 3600-PRO data sheet indicates because the data sheet assumes that the control and shift inputs to the encoder are active when high.* In the Apple, the control and shift inputs to the encoder are active when low.

At the same time that mode ident is latched at B2B3, unique code for the X,Y junction of the matrix key that is pressed is latched at B1B4B5B6B7B8B9. For example, if the X0-Y0 junction key is pressed, 0000000 is latched at B1B4B5B6B7B8B9.

You can represent the X0-Y0 junction as XY = 00 or the X5-Y4 junction as XY = 54 (decimal). When you do this, the B1B4B5B6B7B8B9 code for any keypress is the binary equivalent of the decimal number XY. The 7-digit binary equivalent of 54 is 0110110, so when the X5-Y4 switch is pressed, the 3600-PRO latches 0110110 at B1B4B5B6B7B8B9.

The B1B4B5B6B7B8B9,B2B3 code from the keyboard encoder is applied to the A8A7A6A5A4A3A2, A1A0 address lines of the keyboard ROM. Since B1B4B5B6B7B8B9 is the binary equivalent of XY, the ROM address for any matrix keypress is equal to XY x 4. For example, the "J" key is located at the X2-Y6 junction so the address for "J" in the keyboard ROM is 26 x 4 = 104 = $68. Therefore, the keyboard ROM contains ASCII for SHIFT/CONTROL-J at $68, CONTROL-J at $69, SHIFT-J at $6A, and ALONE-J at $6B.

The two most significant addressing bits of the keyboard ROM, A9 and A10, are connected to the **CAPLOCK'** and **ENVID'** (AN2 if Rev A) lines. What this means is that there are four separate sets of ASCII in the keyboard ROM, two of which are commonly used. The two which are commonly used are the caps locked, standard set ($000—$13F) and the caps unlocked, standard set ($200—$33F). Continuing with the previous example, the caps locked, standard addresses for J are $68—$6B, and the caps unlocked, standard addresses for J are $268—$26B.

The caps locked set is almost identical to the caps unlocked set. The only difference is that the ALONE code for the 26 letters of the alphabet is upper case, not lower case. Thus, the CAPS LOCK key is used to force alphabetic characters to upper case without affecting the numeric and special characters.

*There are internal pull down resistors on the control and shift inputs to the 3600 so there is good reason for the data sheet assumption that these lines are active when high. The internal resistance values are very large, however, and the 1K motherboard resistors pull the SHFT' and CTRL' lines very close to 5 volts when neither SHIFT or CONTROL is pressed.

The two sets of keyboard ASCII which are not commonly used are the caps locked, alternate set ($400—$53F) and the caps unlocked, alternate set ($600—$73F). In American Apples this alternate set contains the Dvorak (American simplified) keyboard layout. Dvorak is a superior keyboard layout which allows faster touch typing than the QWERTY layout, but is rarely used because the QWERTY layout is the English language standard. It is also rarely used in the Apple IIe because many owners don't even know that Dvorak is built in, and they wouldn't know how to access it if they did know it was there. An application note at the end of this chapter shows how to install a switch that will allow the operator to select the standard keyboard layout or the alternate layout.

In Apples built for use in foreign countries, the standard keyboard set is the same as that of American Apples. The alternate set, however, corresponds to the normal keyboard layout for the primary language of that country. Additionally, on export motherboards designed for use in countries using the PAL television system, a simple ALTCHR line selects between the standard and alternate keyboard and video character sets without conflict with the ENVID' line (see Figure 8.6).

Table 7.2 summarizes the contents of the keyboard ROM of American Apples. This table is indexed by XY matrix number, so it should be very useful to persons wishing to program a custom keyboard EPROM. ASCII output for all four keyboard sets and all four mode idents is shown, as well as a base address for each key showing the ROM address of the caps locked, standard, SHIFT/CONTROL entry.

Apple has used two different versions of the keyboard ROM in its American Apple. The earlier Apple IIe's came with the 341(2)-0132-B ROM, and more recent Apple IIe's come with the 341(2)-0132-C ROM (Apple's part numbers)*. There is very little difference between the B and C ROMs—just some minor changes to the Dvorak special characters and the numeric keypad characters. Those matrix junctions in Table 7.2 which have a (B) entry and a (C) entry are those that were changed in the C ROM.

The apparent reason for changing the keyboard ROM was that the quote, colon, and question mark keys in the Apple Dvorak layout were not in accordance with a standard Dvorak layout being proposed by the American National Standards Institute. Since Apple has taken on a leadership role by including Dvorak as an alternate layout, it is good that they quickly took steps to conform to the proposed standard.

The change to the keypad layout looks like an afterthought. The SPACE and ? of the B ROM were deleted to make room for up and down arrows in the C ROM. I assume that Apple makes the C ROM available to anyone who buys a keypad which supports the newer layout. Of course, any keypad manufacturer can sell a keypad with any 4 x 6 or smaller layout and make it work in the Apple IIe by including an inexpensive custom keyboard ROM which the buyer can plug into the motherboard.

Table 7.2 tells a few other interesting tales if you look carefully. For one thing, the Apple IIe keyboard is capable of producing every ASCII code from $0—$7F when used with the keyboard ROM that comes with the computer. CONTROL-2 and CONTROL-6 appear to have been made to correspond to ASCII $00 and $1E specifically to attain this capability.

Another feature of keyboard operation that can be gleaned from Table 7.2 is that, of the special function matrix keys, only ESC and DELETE produce unique code (excluding numeric keypad keys). TAB, RETURN, left arrow, right arrow, down arrow, and up arrow produce code that is identical to that of CONTROL-I, CONTROL-M, CONTROL-H, CONTROL-U, CONTROL-J, and CONTROL-K, respectively.

Keyboard Operational Summary

Circuits related to the keyboard operation have been covered in other chapters and sections of *Understanding the Apple IIe*. This includes the AUTOSTRB circuitry of Figure 3.8, the KBD' generation logic of Figure 5.13b, and the KEYSTROBE soft switch and AKD reading circuitry of Figure 7.1. So the reader will not have to search the entire book for keyboard reference material, a summary of keyboard operational features is presented here.

1. ASCII for any matrix keypress becomes available for reading by the MPU approximately 10 milliseconds after the key is pressed.
2. Apple IIe keys roll over so that if a key is pressed when another key is held down, the ASCII for the new key will be read by a program after the bounce mask period.
3. When an address in the $C000—$C01F range is read, the MMU brings KBD' low during PHASE 0 to gate the ASCII of the last matrix keypress from the keyboard ROM to MD0—MD6 of the data bus.

*Apple documentation and the copyright notice in the keyboard ROM refer to the part number as 341-0132. However, the ROM in my Apple IIe is marked 342-0132-B

4. The AKD line goes high while any matrix key is held down. AKD is routed to the IOU, and the IOU places the state of AKD (delayed 2—3 MPU clock periods) on MD7 of the data bus if $C010 is read. For about 10 milliseconds after a key is pressed, the MPU will sense AKD high at $C010 but will read outdated keyboard ASCII at $C000—$C01F.
5. The KSTRB line goes high for approximately 20 microseconds one time for each matrix keypress. There is a bounce mask period of about 10 milliseconds between the keypress and the KSTRB pulse. The KSTRB pulse occurs about 20 microseconds after the ASCII for that keypress is available for reading by the MPU. KSTRB is routed to the IOU where it is delayed 2—3 MPU clock periods before it sets the KEYSTROBE soft switch.
6. If a key is held down for 534—801 milliseconds (depending on the relationship of the keypress to flash counter bit F3), the IOU starts to generate internal auto repeat strobes at the rate of 15 Hz. These AUTOSTRBs set the KEYSTROBE soft switch as if the operator was pressing a key repeatedly at a very fast rate. Both the AUTOSTRB delay and frequency are functions of the display vertical scan rate (Figure 3.8).
7. Either AKD or KSTRB clears the AUTOSTRB delay generator in the IOU so that pressing any matrix key interrupts the auto repeat function.
8. The KEYSTROBE soft switch is set by KSTRB or AUTOSTRB. It is reset at power up or by MPU access to $C010 or write access to $C01X. Its state is read via MD7 of the data bus when read access is made to $C00X.
9. Programmable functions are:

ADDRESS	FUNCTION
R$C00X	Read KSTRB and ASCII
R$C010	Read AKD and ASCII; reset KEYSTROBE
R$C011—$C01F	Read IOU or MMU soft switch and ASCII
W$C01X	Reset KEYSTROBE

PERIPHERAL SLOT CONNECTIONS

It is hard to know where to start talking about peripheral slot I/O. You can do so much from the slots. They are as versatile as modern microcomputer architecture with full connection to the address bus and data bus. It's like someone designed a really neat computer but on the blueprints drew even empty squares with the message, "user, please fill in the blanks." One never knows what lurks beneath the lid of an innocent looking Apple.

The capabilities of the peripheral slots seem more clear when you look at the connected signals in groups. Figure 7.6 illustrates the peripheral slot connections—grouping the signals functionally. The power supply voltages, for instance, are all grouped together. One quickly sees that all of the **power supply** voltages available in the Apple are also available at the peripheral slots.

Paramount in importance among the peripheral slot signals are the **address bus** with **R/W'**, the **data bus**, and the **timing** inputs. Consider what we know about MPU control of the Apple. All data transfer is over the data bus under control of the address bus during PHASE 0. All I/O control is via the address bus. The correct inference is that you can duplicate any motherboard action with a peripheral card design. This gives you an idea of the variety of tasks that can be accomplished. Of course, most of the things people are going to stuff into Apple peripheral slots haven't even been dreamed of yet.

The peripheral slot data bus is connected to the main data bus through the bidirectional driver, B2. The purpose of this driver is to supply the data bus current needs of a variety of peripheral cards that might be installed. Because of the driver, the load on the main data bus does not fluctuate as the peripheral slot load is varied, and a heavier peripheral slot load can be tolerated than could be without the driver.

The MMU provides directional control to the bidirectional driver via the MD IN/OUT' line. The nature of this control is such that the driver is operationally transparent. In other words, the operation is just about the same as if the peripheral slots were connected directly to the data bus. Simply put, when data needs to be transmitted from the slots or serial input multiplexor to the data bus (such as when a read access is made to a DEVICE SELECT' address), the MMU brings MD IN/OUT' high during PHASE 0 so that direction of the driver is in to the data bus. Otherwise, the MMU leaves MD IN/OUT' low.*

The OE' input to the bidirectional driver is always low except during PHASE 1 of write cycles. Actually, the Apple IIe would work very well if OE' was simply grounded. The only effect I can see of isolating the data bus from the peripheral data bus during PHASE 1 of write cycles is to prevent video data from motherboard RAM from being available at the

*See Chapter 5, **MEMORY MANAGEMENT IN THE APPLE IIe. KDB' and MD IN/OUT'** for a complete discussion of the MD IN/OUT' signal.

Table 7.2 Apple IIe Keyboard ASCII (1 of 2).

XY	SW #	ROM* ADDR	QWERTY	CAPS LOCK BO	CT	SH	AL	NO CAP LOCK BO	CT	SH	AL	DVORAK	CAPS LOCK BO	CT	SH	AL	NO CAP LOCK BO	CT	SH	AL
00	01	$000	ESCAPE	1B	1B	1B	1B	1B	1B	1B	1B	ESCAPE	1B	1B	1B	1B	1B	1B	1B	1B
01	02	$004	1 !	21	31	21	31	21	31	21	31	1 !	21	31	21	31	21	31	21	31
02	03	$008	2 @	00	00	40	32	00	00	40	32	2 @	00	00	40	32	00	00	40	32
03	04	$00C	3 #	23	33	23	33	23	33	23	33	3 #	23	33	23	33	23	33	23	33
04	05	$010	4 $	24	34	24	34	24	34	24	34	4 $	24	34	24	34	24	34	24	34
05	07	$014	6 ^	1E	1E	5E	36	1E	1E	5E	36	6 ^	1E	1E	5E	36	1E	1E	5E	36
06	06	$018	5 %	25	35	25	35	25	35	25	35	5 %	25	35	25	35	25	35	25	35
07	08	$01C	7 &	26	37	26	37	26	37	26	37	7 &	26	37	26	37	26	37	26	37
08	09	$020	8 *	2A	38	2A	38	2A	38	2A	38	8 *	2A	38	2A	38	2A	38	2A	38
09	10	$024	9 (	28	39	28	39	28	39	28	39	9 (	28	39	28	39	28	39	28	39
10	16	$028	TAB	09	09	09	09	09	09	09	09	TAB	09	09	09	09	09	09	09	09
11	17	$02C	Q	11	11	51	51	11	11	51	71	/ ? (B)	3F	2F	3F	2F	3F	2F	3F	2F
11	17	$02C	Q	11	11	51	51	11	11	51	71	' " (C)	22	27	22	27	22	27	22	27
12	18	$030	W	17	17	57	57	17	17	57	77	, <	3C	2C	3C	2C	3C	2C	3C	2C
13	19	$034	E	05	05	45	45	05	05	45	65	. >	3E	2E	3E	2E	3E	2E	3E	2E
14	20	$038	R	12	12	52	52	12	12	52	72	P	10	10	50	50	10	10	50	70
15	22	$03C	Y	19	19	59	59	19	19	59	79	F	06	06	46	46	06	06	46	66
16	21	$040	T	14	14	54	54	14	14	54	74	Y	19	19	59	59	19	19	59	79
17	23	$044	U	15	15	55	55	15	15	55	75	G	07	07	47	47	07	07	47	67
18	24	$048	I	09	09	49	49	09	09	49	69	C	03	03	43	43	03	03	43	63
19	25	$04C	O	0F	0F	4F	4F	0F	0F	4F	6F	R	12	12	52	52	12	12	52	72
20	31	$050	A	01	01	41	41	01	10	41	61	A	01	01	41	41	01	01	41	61
21	33	$054	D	04	04	44	44	04	04	44	64	E	05	05	45	45	05	05	45	65
22	32	$058	S	13	13	53	53	13	13	53	73	O	0F	0F	4F	4F	0F	0F	4F	6F
23	36	$05C	H	08	08	48	48	08	08	48	68	D	04	04	44	44	04	04	44	64
24	34	$060	F	06	06	46	46	06	06	46	66	U	15	15	55	55	15	15	55	75
25	35	$064	G	07	07	47	47	07	07	47	67	I	09	09	49	49	09	09	49	69
26	37	$068	J	0A	0A	4A	4A	0A	0A	4A	6A	H	08	08	48	48	08	08	48	68
27	38	$06C	K	0B	0B	4B	4B	0B	0B	4B	6B	T	14	14	54	54	14	14	54	74
28	40	$070	; :	3A	3B	3A	3B	3A	3B	3A	3B	S	13	13	53	53	13	13	53	73
29	39	$074	L	0C	0C	4C	4C	0C	0C	4C	6C	N	0E	0E	4E	4E	0E	0E	4E	6E
30	44	$078	Z	1A	1A	5A	5A	1A	1A	5A	7A	' " (B)	22	27	22	27	22	27	22	27
30	44	$078	Z	1A	1A	5A	5A	1A	1A	5A	7A	; : (C)	3A	3B	3A	3B	3A	3B	3A	3B
31	45	$07C	X	18	18	58	58	18	18	58	78	Q	11	11	51	51	11	11	51	71
32	46	$080	C	03	03	43	43	03	03	43	63	J	0A	0A	4A	4A	0A	0A	4A	6A
33	47	$084	V	16	16	56	56	16	16	56	76	K	0B	0B	4B	4B	0B	0B	4B	6B
34	48	$088	B	02	02	42	42	02	02	42	62	X	18	18	58	58	18	18	58	78
35	49	$08C	N	0E	0E	4E	4E	0E	0E	4E	6E	B	02	02	42	42	02	02	42	62
36	50	$090	M	0D	0D	4D	4D	0D	0D	4D	6D	M	0D	0D	4D	4D	0D	0D	4D	6D
37	51	$094	, <	3C	2C	3C	2C	3C	2C	3C	2C	W	17	17	57	57	17	17	57	77
38	52	$098	. >	3E	2E	3E	2E	3E	2E	3E	2E	V	16	16	56	56	16	16	56	76
39	53	$09C	/ ?	3F	2F	3F	2F	3F	2F	3F	2F	Z	1A	1A	5A	5A	1A	1A	5A	7A
40	KP	$0A0	/	2F	2F	2F	2F	2F	2F	2F	2F	/	2F	2F	2F	2F	2F	2F	2F	2F
41	KP	$0A4	left(B)	08	08	08	08	08	08	08	08	left(B)	08	08	08	08	08	08	08	08
41	KP	$0A4	down(C)	0A	0A	0A	0A	0A	0A	0A	0A	down(C)	0A	0A	0A	0A	0A	0A	0A	0A
42	KP	$0A8	0	30	30	30	30	30	30	30	30	0	30	30	30	30	30	30	30	30
43	KP	$0AC	1	31	31	31	31	31	31	31	31	1	31	31	31	31	31	31	31	31
44	KP	$0B0	2	32	32	32	32	32	32	32	32	2	32	32	32	32	32	32	32	32
45	KP	$0B4	3	33	33	33	33	33	33	33	33	3	33	33	33	33	33	33	33	33
46	29	$0B8	\ |	1C	1C	7C	5C	1C	1C	7C	5C	\ |	1C	1C	7C	5C	1C	1C	7C	5C

LEGEND: AL = ALONE
SH = SHIFT
CT = CONTROL
BO = BOTH
KP = KEYPAD KEY
(B) = 341(2)-0132-B KEYBOARD ROM
(C) = 341(2)-0132-C KEYBOARD ROM

NOTES: * Add $00n for caps lock, standard
$20n for no lock, standard
$40n for caps lock, Dvorak
$60n for no lock, Dvorak

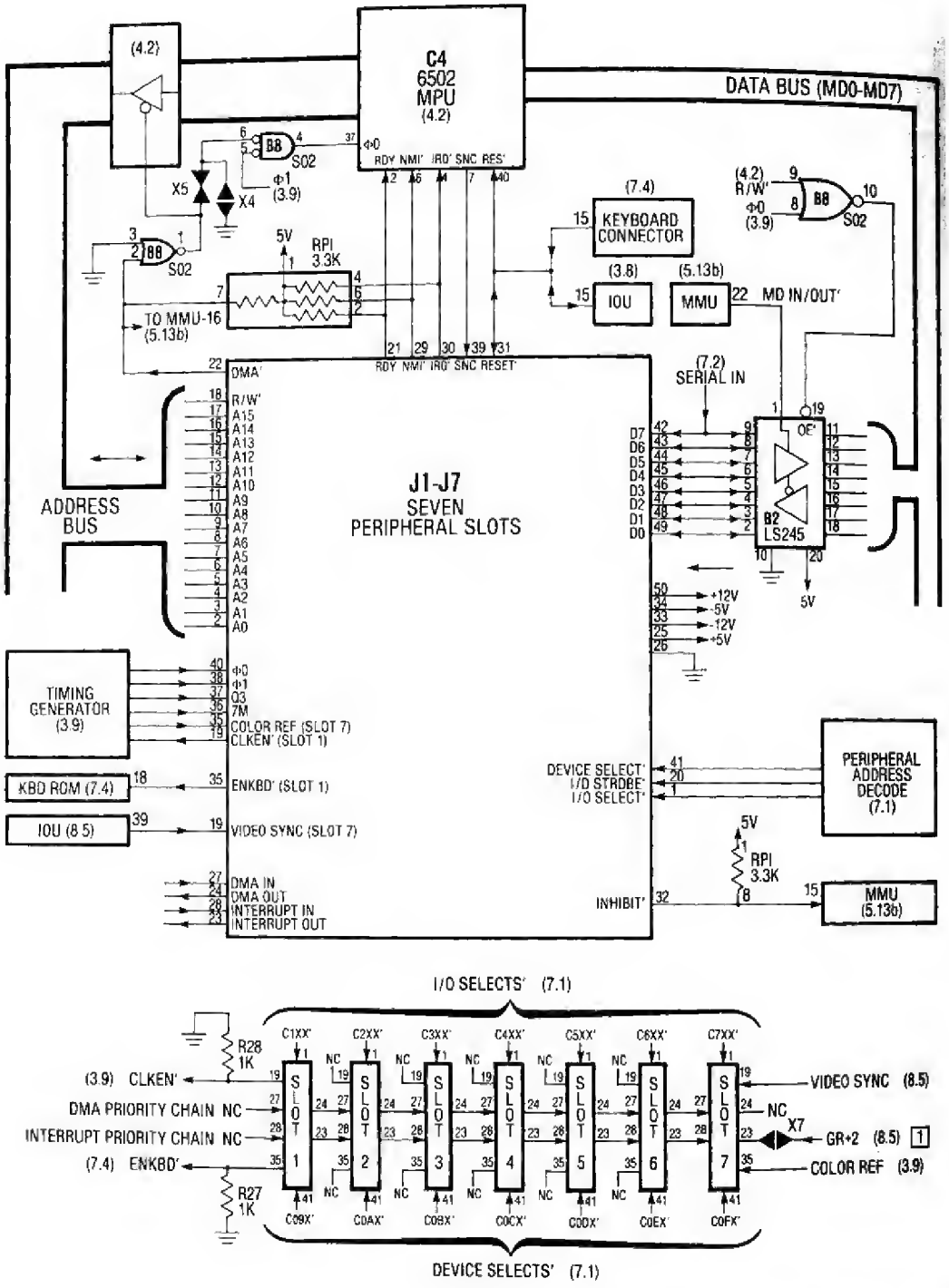
where n = 0 for BOTH
1 for CONTROL
2 for SHIFT
3 for ALONE

Table 7.2 Apple IIe Keyboard ASCII (2 of 2).

XY	SW #	ROM* ADDR	QWERTY	CAPS LOCK BO	CT	SH	AL	NO CAP LOCK BO	CT	SH	AL	DVORAK	CAPS LOCK BO	CT	SH	AL	NO CAP LOCK BO	CT	SH	AL
47	13	$0BC	= +	2B	3D	2B	3D	2B	3D	2B	3D	] }	1D	1D	7D	5D	1D	1D	7D	5D
48	11	$0C0	0)	29	30	29	30	29	30	29	30	0)	29	30	29	30	29	30	29	30
49	12	$0C4	- _	1F	1F	5F	2D	1F	1F	5F	2D	[{	1B	1B	7B	5B	1B	1B	7B	5B
50	KP	$0C8	)	29	29	29	29	29	29	29	29	)	29	29	29	29	29	29	29	29
51	KP	$0CC	ESC (B)	1B	1B	1B	1B	1B	1B	1B	1B	ESC (B)	1B	1B	1B	1B	1B	1B	1B	1B
51	KP	$0CC	up (C)	0B	0B	0B	0B	0B	0B	0B	0B	up (C)	0B	0B	0B	0B	0B	0B	0B	0B
52	KP	$0D0	4	34	34	34	34	34	34	34	34	4	34	34	34	34	34	34	34	34
53	KP	$0D4	5	35	35	35	35	35	35	35	35	5	35	35	35	35	35	35	35	35
54	KP	$0D8	6	36	36	36	36	36	36	36	36	6	36	36	36	36	36	36	36	36
55	KP	$0DC	7	37	37	37	37	37	37	37	37	7	37	37	37	37	37	37	37	37
56	56	$0E0	` ~	7E	60	7E	60	7E	60	7E	60	` ~	7E	60	7E	60	7E	60	7E	60
57	26	$0E4	P	10	10	50	50	10	10	50	70	L	0C	0C	4C	4C	0C	0C	4C	6C
58	27	$0E8	[{	1B	1B	7B	5B	1B	1B	7B	5B	; : (B)	3A	3B	3A	3B	3A	3B	3A	3B
58	27	$0E8	[{	1B	1B	7B	5B	1B	1B	7B	5B	/ ? (C)	3F	2F	3F	2F	3F	2F	3F	2F
59	28	$0EC	] }	1D	1D	7D	5D	1D	1D	7D	5D	= +	2B	3D	2B	3D	2B	3D	2B	3D
60	KP	$0F0	*	2A	2A	2A	2A	2A	2A	2A	2A	*	2A	2A	2A	2A	2A	2A	2A	2A
61	KP	$0F4	rght(B)	15	15	15	15	15	15	15	15	rght(B)	15	15	15	15	15	15	15	15
61	KP	$0F4	left(C)	08	08	08	08	08	08	08	08	left(C)	08	08	08	08	08	08	08	08
62	KP	$0F8	8	38	38	38	38	38	38	38	38	8	38	38	38	38	38	38	38	38
63	KP	$0FC	9	39	39	39	39	39	39	39	39	9	39	39	39	39	39	39	39	39
64	KP	$100	.	2E	2E	2E	2E	2E	2E	2E	2E	.	2E	2E	2E	2E	2E	2E	2E	2E
65	KP	$104	+	2B	2B	2B	2B	2B	2B	2B	2B	+	2B	2B	2B	2B	2B	2B	2B	2B
66	42	$108	RETURN	0D	0D	0D	0D	0D	0D	0D	0D	RETURN	0D	0D	0D	0D	0D	0D	0D	0D
67	63	$10C	up	0B	0B	0B	0B	0B	0B	0B	0B	up	0B	0B	0B	0B	0B	0B	0B	0B
68	58	$110	SPACE	20	20	20	20	20	20	20	20	SPACE	20	20	20	20	20	20	20	20
69	41	$114	' "	22	27	22	27	22	27	22	27	- _	1F	1F	5F	2D	1F	1F	5F	2D
70	KP	$118	? (B)	3F	3F	3F	3F	3F	3F	3F	3F	? (B)	3F	3F	3F	3F	3F	3F	3F	3F
70	KP	$118	ESC (C)	1B	1B	1B	1B	1B	1B	1B	1B	ESC (C)	1B	1B	1B	1B	1B	1B	1B	1B
71	KP	$11C	SPCE(B)	20	20	20	20	20	20	20	20	SPCE(B)	20	20	20	20	20	20	20	20
71	KP	$11C	rght(C)	15	15	15	15	15	15	15	15	rght(C)	15	15	15	15	15	15	15	15
72	KP	$120	(	28	28	28	28	28	28	28	28	(	28	28	28	28	28	28	28	28
73	KP	$124	-	2D	2D	2D	2D	2D	2D	2D	2D	-	2D	2D	2D	2D	2D	2D	2D	2D
74	KP	$128	RETURN	0D	0D	0D	0D	0D	0D	0D	0D	RETURN	0D	0D	0D	D0	0D	0D	0D	0D
75	KP	$12C	,	2C	2C	2C	2C	2C	2C	2C	2C	,	2C	2C	2C	2C	2C	2C	2C	2C
76	14	$130	DELETE	7F	7F	7F	7F	7F	7F	7F	7F	DELETE	7F	7F	7F	7F	7F	7F	7F	7F
77	62	$134	down	0A	0A	0A	0A	0A	0A	0A	0A	down	0A	0A	0A	0A	0A	0A	0A	0A
78	60	$138	left	08	08	08	08	08	08	08	08	left	08	08	08	08	08	08	08	08
79	61	$13C	right	15	15	15	15	15	15	15	15	right	15	15	15	15	15	15	15	15
80	**	$140	J	20	20	20	4A	20	20	20	4A	J	20	20	20	4A	20	20	20	4A
81	**	$144	O	20	20	4F	20	20	20	4F	20	O	20	20	4F	20	20	20	4F	20
82	**	$148	H	20	48	20	20	20	48	20	20	H	20	48	20	20	20	48	20	20
83	**	$14C	N	4E	20	20	20	4E	20	20	20	N	4E	20	20	20	4E	20	20	20
84	**	$150		20	20	20	20	20	20	20	20		20	20	20	20	20	20	20	20
85	**	$154		20	20	20	20	20	20	20	20		20	20	20	20	20	20	20	20
86	**	$158	M	20	20	20	4D	20	20	20	4D	M	20	20	20	4D	20	20	20	4D
87	**	$15C	A	20	20	41	20	20	20	41	20	A	20	20	41	20	20	20	41	20
88	**	$160	C	20	43	20	20	20	43	20	20	C	20	43	20	20	20	43	20	20
89	**	$164	D	44	20	20	20	44	20	20	20	D	44	20	20	20	44	20	20	20

** XY = 80-89 are not used in the Apple IIe because X8 of the keyboard encoder is not connected. In the Apple ROMs, this area contains the name, "JOHN MACD", short for John MacDougall.

*** Addresses $160-$1FF, $360-$3FF, $560-$5FF, and $760-$7FF are not used because the keyboard encoder supports 90 matrix junctions, not 128. In Apple ROMs, this area is filled with $A0, except for $7D8-$7FF which contains this or a similar message: "341-0132B COPYRIGHT APPLE COMPUTER 1982".



NOTES:

[1] Slot 7 pin 23 is not connected in revision A. X7 is present in Revision B only.

Figure 7.6 Schematic: Apple IIe Peripheral Slot Connections.

peripheral slots when PHASE 0 rises during write cycles. I suspect that Apple did this for compatibility with the Apple II although I might be missing something they ran up against in the design.

A number of timing signals are available at the peripheral slots. You can't have too many of these signals at hand to help synchronize peripheral card functions to the motherboard. PHASE 1, Q3, 7M, COLOR REFERENCE (Slot 7), 6502 SYNC, and video SYNC' (Slot 7) are present. Notably absent are 14M, RAS', CAS', and an INHIBIT' priority chain. With signals like these begging to be connected, it's surprising that pins 19 and 35 of Slots 2 through 6 are not connected.

Important **6502 control** inputs are connected to the peripheral slots. These are IRQ', NMI', RESET', and RDY. They are connected in a **wire-OR** configuration with 3300 ohm pull-up resistors so any card can cause an interrupt, reset the Apple, or stop the 6502 via the READY line. The RESET' line is also connected to the keyboard RESET key and to the IOU which responds to RESET' in addition to generating a negative RESET' pulse when the Apple is first turned on.

Other wire-OR lines from the peripheral slots are the DMA' line and the INHIBIT' line. **DMA'** allows a peripheral card to isolate the MPU from the address bus and data bus so it can gain control of the Apple for fast I/O or other purpose. **INHIBIT'** disables MPU communication with motherboard and auxiliary card memory, and opens up $0000—$BFFF and $D000—$FFFF addressing for any sort of peripheral card response.* Note that between INHIBIT', INTCXROM, SLOTC3ROM, and the Slot 1 ENKBD' line, provisions exist for assigning all addressing in the $0000—$C01F and $C090—$FFFF ranges to the peripheral slots.

The Apple IIe is different from the Apple II in that 3300 ohm resistors are used to pull up the peripheral slot wire-OR lines as opposed to the 1000 ohm pull-up resistors of the Apple II. This value may have been switched because the 6502 data sheet specifies 3000 ohm resistors for 6502 wire-OR lines, but it sure lengthens the rise time of the wire-OR lines. In fact, the 3300 ohm resistors cause such slow switching that some peripheral cards may require a parallel pull-up resistor on a wire-OR line to operate properly in the Apple IIe. I found that this was necessary on the DMA' line to enable D MAnual Controller (Figure 4.7) to successfully steal a cycle from the old DMA based *Softcard* in an Apple IIe.

Slot 1 has two signals connected which are not available at the other slots, **ENKBD'** and **CLKEN'**. These are present primarily for diagnostic and testing purposes, but innovative peripheral card designs could make good use of them. When ENKBD' is high, the keyboard ROM is inhibited and the other devices can place data on MD0—6 of the data bus when read access is made to $C000—$C01F. When the CLKEN' is high, the motherboard 14M signal is disabled, and a card in the auxiliary slot can inject an alternate master clockpulse reference onto the 14M line. If, as normally is the case, ENKBD' and CLKEN' are not connected on the Slot 1 card, motherboard pull-down resistors pull these lines low so the affected circuits can function normally.

The USER1' line of the Apple II is not present in the Apple IIe. This was a hardware means by which any peripheral card could disable all I/O address decoding in the $C000—CFFF range by pulling a single line low. In the Apple IIe, motherboard or peripheral card resident programs can disable I/O decoding in the $C100—$CFFF range using the INTCXROM, SLOTC3ROM, and INTC8ROM soft switches, but the capability of inhibiting DEVICE SELECT' and other $C0XX decoding does not exist.

Pin 39, the USER1 line in the Apple II, is the **6502 SYNC** line in the Apple IIe. Having 6502 SYNC present is very helpful in hardware applications such as execution of single instructions and identifying op codes on the data bus. For example, it is possible for a peripheral card in any Apple IIe slot to detect the execution of an RTI instruction by looking for SYNC high and $40 on the data bus when PHASE 0 falls. By detecting 6502 instructions in this manner, it is actually possible for peripheral cards to respond directly to 6502 program instructions.

The DEVICE SELECT's, I/O SELECT's, and I/O STROBE' are address decoded signals which identify addresses on the address bus in the ranges assigned to the peripheral slots. These address ranges could have been assigned by convention only. For example, a Slot 1 peripheral card could easily decode the $C09X address range without the aid of its DEVICE SELECT' input, but then how could you operate that card in a slot other than Slot 1? The card would have to have switches to configure it for different slots. Also, having DEVICE SELECT' decoded on the motherboard lends the force of hardware reality to the convention that $C09X belongs to Slot 1. This is fairly important considering the diversity of sources for Apple peripheral

INHIBIT' also disables $C100—$CFFF motherboard ROM, so ny peripheral card can use INHIBIT' to gain access to $C100—CFFF addresses that are configured for motherboard ROM esponse.

cards. Needless to say, decoding the address ranges on the motherboard also reduces the chip count of most peripheral cards.

The **DEVICE SELECT'** input to each peripheral slot identifies a 16-bit address range assigned specifically to that slot. This range is normally used to command a peripheral to do things like gate data to the data bus or disable one of its functions. A card design may require only one programmed command, such as a speech synthesis board which says a word when you store a value to its only address. This type of card can use the DEVICE SELECT' to trigger its action and it requires no on-board address decoding circuitry. Peripheral slot RAM cards usually have a number of different configurations. DEVICE SELECT' enables reconfiguation of the card, and the new configuration is determined by the low order address bits and R/W'. A card can distinguish between 32 possible commands in the DEVICE SELECT' range by decoding the states of A0, A1, A2, A3, and R/W'.

The **I/O SELECT'** input to each peripheral slot identifies a 256-byte addressing range uniquely assigned to that slot. This address range is normally used by a 256-byte program in ROM or PROM. It has to be taken up by a 256-byte program if the card is to be capable of response to BASIC "**PR#**" and "**IN#**" commands. What these commands do is cause program flow to vector to the first address of the I/O SELECT' range, so there must be a program stored there if PR# and IN# are going to work.*

The **I/O STROBE'** signal identifies $C800—$CFFF addressing for all seven peripheral slots which share this range on an equal basis. The idea is to store a big I/O handling program on a 2K ROM or PROM on a peripheral card and then give that card sole access to $C800—$CFFF when it is active for input or output. As described in the next section, the peripheral cards utilizing this "**I/O STROBE' ROM**" must deactivate response to the I/O STROBE' when $CFFF is detected on the data bus during PHASE 0. This protocol prevents two ROMs from simultaneously trying to control the data bus when I/O STROBE' goes low. The I/O STROBE' ROM capability makes possible such peripherals as smart printers, smart 80-column cards, and smart EPROM programmers with driving programs stored in firmware.

It is important to remember that any program can disable the I/O SELECT' signals and I/O STROBE' by manipulating the INTCXROM, SLOTC3ROM, and INTC8ROM soft switches. Generally, programs will do this because they need to access motherboard firmware routines in the $C100—$CFFF range without interference from peripheral card ROM. The monitor does this in a number of situations. The idea is to disable slot ROM, call the motherboard subroutine, then reenable slot ROM after subroutine execution. Another reason to disable I/O SELECT' and I/O STROBE' would be to enable a peripheral card to steal $C100—$CFFF addressing via the INHIBIT' line.

The remaining four peripheral signals are DMA IN, DMA OUT, INTERRUPT IN, and INTERRUPT OUT. These are the DMA and interrupt **priority chain** connections described in great detail in Chapter 4. They are there to keep two or more cards from trying to simultaneously perform similar functions. Only one card can perform DMA or interrupt the MPU at one time. The priority chains can be used to keep order by giving high priority to lower numbered slots.

A peripheral slot priority chain which does not exist but is very badly needed is an INHIBIT' priority chain. Apple realized this when they designed the old 12K firmware card for the Apple II. Their solution was to use the DMA priority chain to prioritize the INHIBIT' based firmware card so that you can install two or more firmware cards in adjacent slots. If two or more firmware cards are enabled at the same time, only the highest priority card will attempt to respond to its addressing range. This use of the DMA priority chain illustrates that the two priority chains are assigned to the DMA and interrupt functions by convention only. The hardware reality is that you can use the DMA and interrupt priority chains for any sort of serial communication between slots.

There is no need for DMA IN or INTERRUPT IN signals at Slot 1 because Slot 1 is the highest priority slot. Similarly, there is no need for DMA OUT or INTERRUPT OUT signals at Slot 7. As a result, pins 27 and 28 of Slot 1 and pin 24 of Slot 7 are not connected, not assigned, and available for connection to signals if Apple ever decides to do so. Pin 23 of Slot 7 is connected via the X7 jumper to GR+2 from pin 2 of the IOU. If your Slot 7 peripheral requires identification of GRAPHICS time, its installation procedure probably includes soldering of the X7 jumper.

*PR#0, IN#0, PR#3, and IN#3 are interpreted as special cases by Apple IIe firmware. PR#0 and IN#0 cause program flow to vector to motherboard video output and keyboard input routines rather than to a nonexistent program in nonexistent Slot 0, and, if a RAM card is installed in the auxiliary slot, PR#3 and IN#3 cause program flow to vector to motherboard 80-column routines rather than Slot 3 firmware routines.

I/O STROBE' Protocol

The I/O STROBE' signal at pin 20 of the peripheral slots is the ROM enabling signal for any peripheral card on which $C800—$CFFF is currently active. There are no motherboard control signals to tell the various slots when they may or may not respond to the I/O STROBE', so Apple decided on the following protocol which cards responding to I/O STROBE' must follow:

1. When pin 1 (I/O SELECT') goes low, a peripheral card may begin to actively respond to the I/O STROBE' at pin 20.
2. When $CFFF is on the address bus during PHASE 0, all peripheral cards must stop responding to the I/O STROBE'.

An example should clarify how the I/O STROBE' protocol works. Assume that Slot 1 and Slot 2 have peripheral cards installed and that both have a $C800—$CFFF ROM on board. A PR#1 is executed from BASIC which results in 6502 program flow vectoring to address $C100 for all character output. The card at Slot 1 responds to $C1XX addresses by placing a program on the data bus, and by activating response to the I/O STROBE'. Suppose that the program driven out at address $C100 begins with:

```
C1ØØ:  BIT $CFFF
C1Ø3:  JMP $C8ØØ
```

On the last cycle of execution of the first instruction, response to the I/O STROBE' is deactivated on all peripheral cards including Slot 1. However, on the first cycle of execution of the second instruction, $C1XX is back on the address bus and I/O STROBE' response is again activated at Slot 1. It is now safe to begin execution of programs stored in the $C800—$CFFF ROM at Slot 1. The ROM at Slot 2 will not interfere with Slot 1 access to $C800—$CFFF, because it is designed to ignore the I/O STROBE' after an access to $CFFF.

It is possible for a peripheral card to store its $CnXX program and its $C800—$CFFF program on a single 2K ROM. This is accomplished by enabling the output of the ROM to the data bus when I/O SELECT' (pin 1) goes low, or when I/O STROBE' response is activated and I/O STROBE' (pin 20) goes low. If the card is located at Slot 1, the 256 bytes at $C1XX will be identical to the 256 bytes at $C9XX. The 256 bytes at $C1XX can be accessed at any time. The 2048 bytes at $C800—$CFFF can be accessed only when I/O STROBE' response is activated (after a $C1XX access).

Recall from Chapter 5 that INTC8ROM inhibits I/O STROBE' and activates motherboard ROM response to $C800—$CFFF addressing, that INTC8ROM is set by access to $C3XX when SLOTC3ROM is reset, and that INTC8ROM is reset by access to $CFFF. In other words, the Apple IIe 80-column motherboard firmware, in conjunction with SLOTC3ROM and INTC8ROM, fully emulates a Slot 3 peripheral card that responds to I/O SELECT' and I/O STROBE', with one subtle difference. The difference is that motherboard firmware has hardware priority over the slots in response to $C800—$CFFF. When INTC8ROM is set, motherboard firmware doesn't have to make an access to $CFFF to disable peripheral card response to I/O STROBE' because I/O STROBE' is inhibited by INTC8ROM.

The I/O STROBE' ROM capability should not be confused with the capability to steal addresses $C100—$FFFF from motherboard ROM via the INHIBIT' line. I/O STROBE' gives a peripheral card access to 2048 bytes of addressing when INTCXROM and INTC8ROM are reset. By pulling INHIBIT' low, any peripheral card can disable all motherboard memory from response to $0000—$BFFF and $C100—$FFFF.

THE APPLE I/O SYSTEM: KSW AND CSW

The peripheral slot capabilities are determined by the signals connected to them, but our perception of how they work is very much colored by the operating systems that normally control them. The precedents for I/O control in the Apple were established by the old **Monitor ROM**, the 2K ROM which contained $F800—$FFFF firmware in the original Apple II. The main precedent is that memory locations $36 and $37 always contain the address of the Apple's primary output routine, and locations $38 and $39 always contain the address of the Apple's primary input routine. $36 and $37 are referred to in the monitor listing as **CSW** (Character output SWitch), and $38 and $39 are referred to as **KSW** (Keyboard input SWitch).

Apple Monitor I/O

The Apple II/IIe monitor has evolved over the years just as the original Apple II evolved into the Apple IIe, but a very large portion of the Apple IIe monitor is identical to the original monitor that was written by a computer hobbyist in his spare time back in the 70s. In particular, the same system is normally in effect where every keyboard input is followed immediately by video output, and the input

and output routines are determined by KSW and CSW. To see how this system works, let's examine what happens at power up, initially assuming that there is no disk controller installed.

The way the Apple presents itself to us is this: at power up, KSW is set to the address of a firmware routine **(KEYIN)** which waits for a keypress while it displays a cursor; CSW is set to the address of a firmware routine **(COUT1)** which stores the accumulator in TEXT memory while keeping track of the next screen memory address. Then, after searching for a disk controller and not finding it, the Applesoft BASIC interpreter is entered. This program, as do Integer BASIC, the system monitor, and many others, talks to humans through the **GETLN** (GET LiNe) routine.

GETLN gets a series of characters from the primary input device which it finds by doing a JUMP INDIRECT to KSWL ($38). After it gets each character, it stores it in an **Input Buffer** (memory locations $200—$2FF), and sends it to the primary output device by doing a JUMP INDIRECT to CSWL ($36). GETLN continues to input and output characters until it receives a carriage return code from the input device. The program which called GETLN is then able to examine the "line" of data in the input buffer and take action based on its contents. The GETLN routine is largely responsible for our impression of how the Apple talks to us.

Linking I/O to Other Devices

CSW and KSW are the **I/O links**. You can link the driving program for any device to the Apple and make it the primary input or output device. This is because most programs perform input or output by jumping to the address contained in KSW or CSW. As an example, you can connect a serial output device to one of the annunciator ports and place a control program in RAM. You then make this device the Apple's primary output by placing the entry address of your control program at CSW. If your program is typical, after it outputs each character to your device, it jumps to the COUT1 routine so the character is also output to the screen.

Any peripheral slot can be assigned as the Apple's primary input or output device by doing a "PR#n" or "IN#n" from BASIC, or an "n CONTROL-P" or "n CONTROL-K" from the monitor. When a PR#1 is performed, $00 and $C1 are stored at locations $36 and $37. This means that if you do a PR#1, the card in Slot 1 had better respond to $C100 with a program, because the 6502 is going to be executing at that address real soon.

If, at power up, a disk controller is located, the program beginning at $Cn00 ($C600 if Slot 6) on the controller is executed. This is the bootstrap program that begins the DOS loading procedure. After DOS 3.3 is booted, CSW and KSW are set to addresses $9FBD and $9E81 ($B84B and $B84E if ProDOS). Then all input/output passes through the DOS, which checks to see if it is disk related. For example, "CATALOG" is not a valid BASIC command, but it can be executed from BASIC while the DOS is connected, because it is a valid DOS command. While you are entering BASIC code from the keyboard with DOS connected, entries are checked for DOS validity before control is passed to the BASIC interpreter for command processing. If a BASIC program is actually running, the DOS does little processing of input or output data except to check output data for a leading **"CONTROL-D"** character. The "CONTROL-D" is a flag which tells the DOS that a disk related command follows.

Entering PR#2 while the DOS is connected does not make Slot 2 the primary output slot. CSW and KSW will still contain $9EBD and $9E81. The DOS intercepts the PR#2 and does its own output setting routine. DOS maintains its own I/O links—we will call them DOSKSW and DOSCSW. DOSCSW is $AA53,$AA54 ($BE30/$BE31 if ProDOS), and DOSKSW is $AA55/$AA56 ($BE32/$BE33 if ProDOS). PR#2 with DOS 3.3 connected results in $AA53 and $AA54 being set to $00 and $C2. Slot 2 becomes the secondary output behind the DOS. If a PR#2 is performed from a running program, Slot 2 will probably actually become the primary input and output device, or it may not become connected at all, depending on its $C2XX firmware. If the $C2XX firmware automatically sets both CSW and KSW to some value, then the DOS will be disconnected and Slot 2 will be connected as primary input and output. If the $C2XX firmware leaves the DOS connected at KSW, the first time an input is performed, the DOS will disconnect Slot 2 and reset CSW to $9EBD. The way to do a PR#2 from a running BASIC program and leave the DOS connected is to do a PRINT : PRINT CHR$(4); "PR#2". The CHR$(4), CONTROL-D, flags the DOS that a disk related command is following. The DOS type PR#2 is performed, making Slot 2 the secondary output behind the DOS.*

*A subtle difference between DOS 3.3 and ProDOS processing is that DOS 3.3 identifies disk related commands from a carriage return output followed by "CONTROL-D", while ProDOS identifies disk related commands from a BASIC PRINT statement in which "CONTROL-D" is the first character output. PRINT : PRINT CHR$(4);"PR#2" should satisfy the requirements of both DOS 3.3 and ProDOS.

Similar steps must be taken in assembly language programs. If you change CSW to $9000, then the first time a character input is called for, the DOS will change CSW back to $9EBD. You can do one of three things to get around this. You can change CSW to $9000 and change KSW to $FD1B, disconnecting the DOS entirely and connecting the keyboard as primary input. You can modify DOSCSW (AA53/4) to $9000, leaving the DOS connected. You can also store $9000 at CSW and do a "JSR $3EA". This is a DOS routine which takes the value you stored at CSW or KSW and transfers it to DOSCSW or DOSKSW, then restores CSW and KSW to $9EBD and $9E81. $3EA is easy to remember if you remember "3 EACH".*

Peripheral Cards and Primary I/O Devices

The various peripheral cards can be divided into three categories: those with onboard firmware at $CnXX, those capable of being the Apple's primary input or output device which have no $CnXX firmware, and those which would normally not be the Apple's primary input or output device. The first category includes such cards as 80-column cards, smart printer interfaces, remote terminal interfaces, and the disk controller. The presence of onboard firmware with response to the simple PR#n and IN#n commands should be an important factor in an Apple owner's choice among similar commercial products.

Peripheral cards with onboard firmware at $CnXX generally will not work in Slot 3 if a RAM card is installed in the auxiliary slot. This is because PR#3 is interpreted by Apple IIe firmware as a command to activate the 80-column firmware if an auxiliary RAM card is present. The basic conflict is that the 80-column firmware is addressed at $C3XX and $C800—$CFFF, the Slot 3 I/O SELECT' and I/O STROBE' ranges. When the 80-column firmware is active, SLOTC3ROM is reset and Slot 3 I/O SELECT' is consequently inhibited. Cards which do not respond to I/O SELECT' have no conflict with the 80-column firmware, and they can be used in Slot 3 without restriction.

The 80-column firmware, in conjunction with SLOTC3ROM and INTC8ROM, emulates a Slot 3 peripheral card that responds to I/O SELECT' and I/O STROBE'. To me, this represents a weakness since the 80-column capability should be automatic and not preclude the use of any I/O device. I don't think you should have to enter PR#3 to get an 80-column display, and I don't think you should lose your 80-column display when you press RESET or enable an I/O device via PR#n. For that matter, I don't think you should have to install an auxiliary card to achieve an 80-column display capability.

The second category of cards is like the first except that the user must load the driving software from disk or other medium. This driving software will typically bury itself above "HIMEM" and link itself to the Apple via CSW and KSW or DOSCSW and DOSKSW. This is a definite step down in convenience from smart cards with firmware at $CnXX and possibly $C800—$CFFF. The program at $CnXX goes a little beyond offering the convenience of PR#n and IN#n commands. Commercial programs such as word processors, assemblers, and data base managers allow records to be output to any slot, if the slot has a $CnXX driver. These programs usually make no provision for linking output to a RAM address.

The third category of cards is not normally linked to the Apple via KSW and CSW. A 16K RAM card is not a conventional I/O device but simple memory expansion. A 128K card, however, may come with an associated disk emulator program which does get linked. The DMA based manual controller shown in Figure 4.7 is a device which would not be thought of in connection with the links. A secondary MPU card would not be linked. A speech synthesis card might be linked, but it would just as often be driven by special purpose subroutines in a larger program.

Understanding which of the three categories the cards in a given Apple fall into is a big step in understanding what is going on in that Apple. The concept of peripheral slots integrated with the bus structure of the motherboard is so powerful that the "spirit" of the Apple may be under the control of any card or associated control program. When the control breaks down and things do not function as they should, the owner has only his own intellect to fall back on to sort things out. Know your peripheral cards; know your motherboard; know your operating systems; know your Apple.

I/O TIMING

I/O timing is the timing of the address decoded signals. All motherboard I/O and most peripheral slot I/O is controlled by signals decoded in the MMU, peripheral address decoding circuitry, or

*There is no equivalent to a "JSR $3EA" in ProDOS, but the equivalents to DOSCSW and DOSKSW do exist. The ProDOS I/O links are VECTOUT ($BE30/$BE31) and VECTIN ($BE32/$BE33), and these locations must be modified directly to link I/O routines to ProDOS.

IOU. Access to a DEVICE SELECT' address exercises much of the address decoding chain, so that is the example chosen for analysis.

Figure 7.7 shows read and write timing for access to $C090. The read and write are identical except for data bus management. Data bus management in a write to $C090 is identical to that of RAM in a write cycle, because in all non DMA write cycles, the data bus and peripheral data bus are receivers of 6502 write data. I/O read cycle data bus management is different from that of a RAM read cycle, because MD IN/OUT' rises during PHASE 0 when I/O addresses are read.* MD IN/OUT' switches the direction of the peripheral data bus driver to allow the MPU to receive data coming from I/O devices.

A description of important events of Figure 7.7 follows here. Please refer also to Figure 7.1 for clarification.

1. CASEN' and CXXX go high after an address in the CXXX range becomes valid on the address bus. Even though these signals are not gated by PHASE 0 in the MMU, the logic circuits which they enable are gated by PHASE 0 (or PHASE 1 low). CASEN' high disables communication with motherboard RAM during PHASE 0, and CXXX high enables I/O signals to be activated during PHASE 0.
2. C0XX' falls after PHASE 0 rises and rises after PHASE 0 falls during access to $C0XX. This enables further decoding in the IOU and the 4 to 16 decoder at C10. I/O SELECT' signal timing is identical to that of C0XX'.
3. C09X' (Slot 1 DEVICE SELECT') falls after C0XX' falls and rises after PHASE 0 falls during access to $C09X. Timing of C04X', C06X', C07X', and the other DEVICE SELECT' signals is identical to that of C09X'.
4. The peripheral data bus driver is isolated from both the data bus and peripheral data bus during PHASE 1 of write cycles. Video data from motherboard RAM is therefore not available at the peripheral slots when PHASE 0 rises during write cycles. During PHASE 0 of write cycles, first motherboard RAM video data, then 6502 write data is passed from the data bus to the peripheral data bus.
5. MD IN/OUT' rises at PHASE 0 rising plus MMU propagation delay (about 60 nsec) during read access to $C090. This switches the direction of peripheral slot data bus driver so that the data bus receives signal information from the peripheral data bus. The peripheral card in Slot 1 can now place data on the peripheral data bus in response to its DEVICE SELECT' input. If the Slot 1 card does not take control, the floating peripheral data bus stores motherboard video data which is valid on the peripheral data bus when MD IN/OUT' rises. The video data is transmitted to the data bus for reading by the MPU.
6. MD IN/OUT' falls at PHASE 0 falling plus MMU propagation delay (about 50 nsec) during read access to $C090. This switches the direction of the peripheral slot data bus driver so that the peripheral data bus receives signal information from the data bus. The Slot 1 card must now release control of the peripheral data bus or it will compete with the peripheral data bus driver. In most, and perhaps all, Apple IIe's, MD IN/OUT' falls after 6502 PHASE 2 falls.

The DEVICE SELECT' and I/O SELECT' signals are commonly used by peripheral cards to gate data to the data bus during read access. Their timing for this function is less than perfect, as Figure 7.7 shows, but they still work. The problem is that DEVICE SELECT' and I/O SELECT' (same as C0XX' in Figure 7.7) come too early. They fall before MD IN/OUT' rises, and they rise before 6502 PHASE 2 falls. As a result, peripheral cards are likely to compete with the peripheral slot driver for control of the peripheral data bus for a short period at the beginning of DEVICE SELECT' or I/O SELECT'. Also, peripheral cards often will not hold read data valid when 6502 PHASE 2 falls.

Early DEVICE SELECT' and I/O SELECT' timing only poses a problem when relatively fast devices (such as the data register of the disk controller) are responding to them with data. With slower devices like NMOS ROM, data response is delayed enough from gating inputs that data is always valid at the right times. But even fast response peripheral cards work with the early DEVICE SELECT' and I/O SELECT'. The short duration bus fight occurs at a time when critical data transfer is not taking place, and it seems to cause no damaging system noise. The early removal of data at the trailing edge causes no problem because data bleeds off very slowly from the floating peripheral data bus. In other words, Apple gets away with it, but peripheral card designers should be aware of the early timing.

*Reading keyboard data is an exception. Bus management when reading keyboard data is identical to that of reading motherboard ROM. In both instances, an MMU signal (KBD', ROMEN1', or ROMEN2') drops low during PHASE 0 to gate data from a ROM to the data bus.

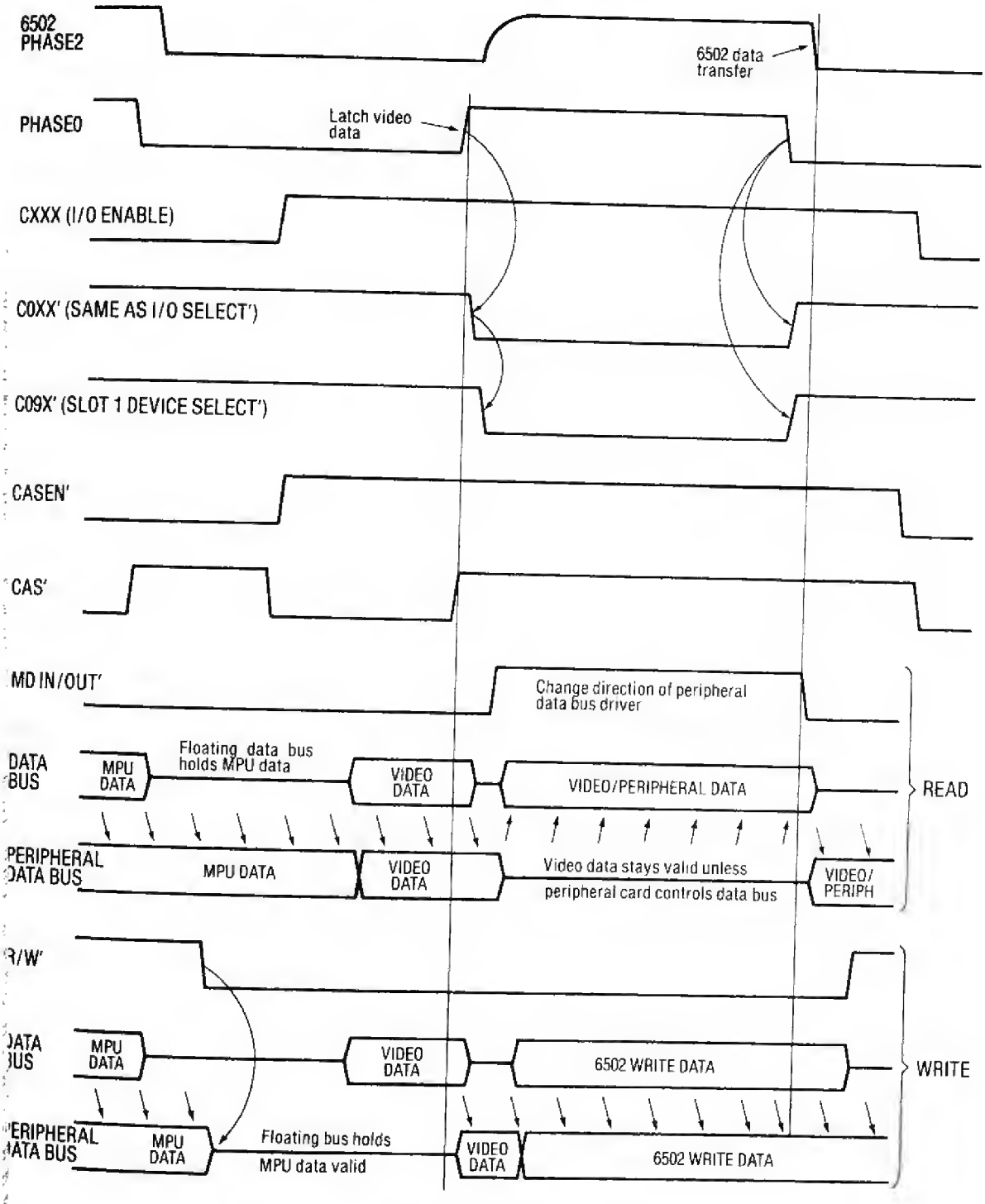


Figure 7.7 Timing Diagram: Read/Write Access to $C090.

Timing for the C06X' serial input enable signal is identical to that of DEVICE SELECT', and the same timing abnormality exists. When read access is made to $C06X, C06X' falls before MD IN/OUT' rises, and the serial input multiplexor momentarily competes with the peripheral driver for control of D7 of the peripheral data bus. Also, C06X' rises before PHASE 2 falls so the serial input multiplexor is not holding D7 valid when PHASE 2 falls. Again, the slow bleed off of data from floating D7 results in correct data transfer to the MPU.

Figure 7.7 shows that when a peripheral card does not respond to read access in its DEVICE SELECT' range by placing data on the data bus, the MPU will read the motherboard video data from PHASE 1 video scanning. This occurs because first the floating data bus, then the floating peripheral data bus, then the floating data bus again store the video data while MD IN/OUT' switches peripheral driver direction to, from, and to the data bus. In the same way, motherboard video data is read by the MPU when access is made to any non-responding address in the $C020—$CFFF range.

THE AUXILIARY SLOT

The auxiliary slot is of a different nature than the peripheral slots. It is not connected to the address bus or 6502 control lines, has no enabling signals like DEVICE SELECT', is not assigned to any addresses in the Apple memory map, and has only a very limited DMA capability. Instead of being a versatile I/O or expansion port in the Apple memory map, the auxiliary slot is wired so that the installed card can be integrated into the operation of a functional area of the Apple IIe. Figure 7.8 shows the wiring connections to the auxiliary card with related signals grouped together. There are three major signal groups, the RAM group, the timing group, and the video generation group.

The **RAM signal group** consists of the multiplexed RAM address bus, the video data bus, the data bus, R/W', R/W'80, CASEN', and EN80'. As shown in Chapter 5, these connections (along with some of the timing inputs) support a 64K RAM card, scanned for video output during PHASE 1, and accessible by the MPU during PHASE 0. This is the function we normally think of in connection with the auxiliary slot, simply because it is the one commonly supported by commercially available cards.

The Apple IIe design supports full access to 64K of RAM in the auxiliary slot, but it is possible to place multiple 64K banks on an auxiliary card and switch between them by decoding the $C07X range. The C07X' signal appears to be connected to pin 6 of the auxiliary slot for just this purpose. In this sense, C07X' is like a DEVICE SELECT' signal for the auxiliary slot. It can be used for this purpose as long as a program that resets the paddle timers does not interfere with the auxiliary card bank selection functions.

Another scheme which has been used in auxiliary card designs is to include an alternate microprocessor in addition to one or more 64K banks of RAM. This means that the auxiliary card is actually a separate microcomputer with its own RAM, coprocessing with the motherboard 6502, and capable of communicating via data in auxiliary RAM accessible by either microprocessor. All I/O in this coprocessing arrangement is accomplished by the motherboard MPU, including loading of programs from disk to auxiliary RAM where they can be executed by the alternate MPU.

The **timing signal group** consists of full connection to all outputs of the timing generator plus the ENTMG' line. A diagnostic card in the auxiliary slot can thus monitor all of the timing signals to determine whether or not they are operating correctly. The diagnostic card can also pull ENTMG' high to disable all outputs of the timing HAL and inject substitute signals to the motherboard (see Figure 3.9). Furthermore, if an associated card in peripheral Slot 1 pulls CLKEN' high to disable 14M, the auxiliary card can inject an alternate 14M signal to the motherboard.

The **video generation signal group** consists of all of the address and chip enable inputs to the **video ROM** plus PICTURE', SYNC', CLRGATE' and ALTVID'.* A card in the auxiliary slot can thus monitor all of the video ROM inputs plus PICTURE', SYNC', and CLRGATE' to verify correct operation and isolate faults. It can also disable the video ROM and, consequently, the PICTURE' signal and then inject its own ALTVID' signal in reaction to the other video group signals and in place of the PICTURE' signal. The ALTVID' signal is therefore an alternate picture signal, injected from the auxiliary card to the motherboard when ENVID' is pulled high. It could also be used to bring various areas of the screen to the white level while ENVID' is low, regardless of the current state of the scanned display map.

*Another way of stating this is that the video generation signal group consists of the IOU signal outputs related to video generation plus PICTURE', ENVID', and the video data bus. Note that the video data bus can be considered as part of the video generation group as well as the RAM group.

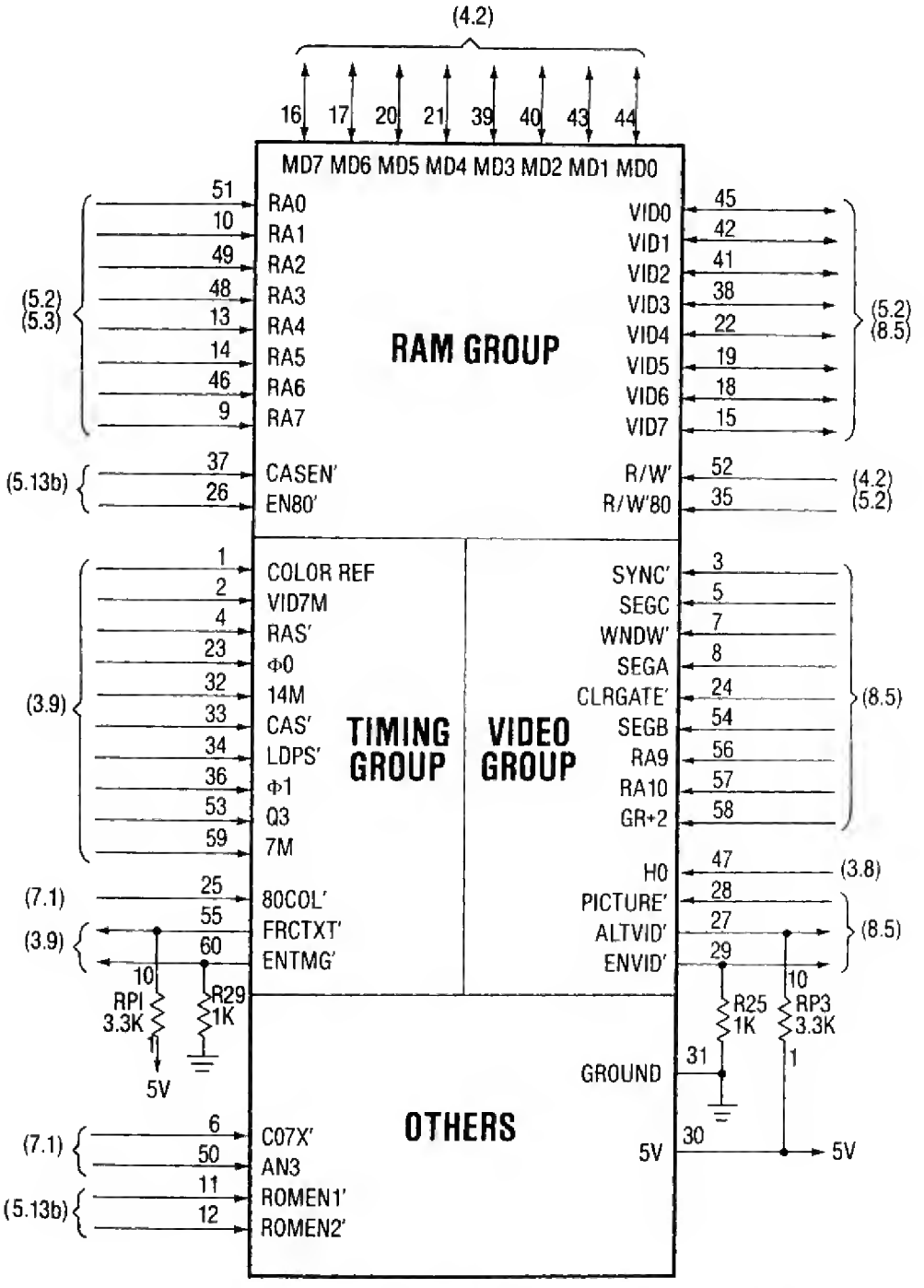


Figure 7.8 Schematic: Apple IIe Auxiliary Slot Connections.

Both the timing and video generation signal groups represent diagnostic capabilities. In other words, these signal groups support computerized verification, fault isolation, and production checkout of Apple IIe motherboards by special purpose auxiliary cards. Of course, it is quite possible that operational designs will appear which utilize alternate timing signal or picture signal injection to achieve a goal, but these cards will probably also include one or more 64K banks of RAM. Too many people like 128K of RAM in their Apple to give up on the idea.

Some auxiliary slot signals do not fit neatly into the three signal groups. The signal groups are not absolute divisions but convenient ways of picturing the auxiliary slot functions. ROMEN1', ROMEN2', EN80', and CASEN' could be thought of as an MMU group, and it is possible that Apple diagnostic cards are used to verify MMU management functions. Also, ROMEN1', ROMEN2', and ENFIRM might be considered as a ROM signal group for Revision A motherboards. An auxiliary card can inhibit Revision A motherboard ROM by pulling ENFIRM low (see Figure 6.1). Whatever Apple had in mind for ENFIRM, they dropped it and changed ENFIRM to FRCTXT' in Revision B (see Figure 3.9). This is the major operational improvement of Revision B, the capability of forcing TEXT mode at the timing generator so that DOUBLE-RES graphics displays are possible.

SOFTWARE APPLICATION

PROGRAMMING THE GAME PADDLES

The PREAD routine of the monitor is pointed out by the *Apple II Reference Manual for IIe Only* as a convenient subroutine for reading any of the four paddle inputs to the Apple. Actually, PREAD is used by the Applesoft and Integer BASIC PDL(n) expressions, and it is called by many programs you might purchase for the Apple. There are some limitations to PREAD which are not irreversible limitations of the Apple. They're just weaknesses in PREAD. This application note explores the PREAD routine and illustrates some alternate programming methods for reading the timers.

PREAD is designed to read the paddle whose number (0—3) is contained in the X-register. For instance, if you want to read Paddle 1, you place 1 in the X-register and do a "JSR $FB1E". PREAD will return with the Accumulator scrambled and a number from 0 to 255 in the Y-register which represents the position of the paddle. The way PREAD works is this:

1. It begins by triggering the four timers (LDA $C070).
2. After 10 cycles, it begins polling the pertinent timer ($C064,X) in an 11-cycle loop. The Y-register is incremented in the loop and thus accumulates the number of loop executions. Program flow exits from PREAD when the pertinent timer is found to be reset.
3. If the polling loop is executed 256 times, the routine is exited immediately with 255 in the Y-register.

The PREAD comment in the monitor listing says "COUNT Y-REG EVERY 12 USEC". This is not true; the Y-register counts approximately every 11 microseconds. Possibly the programmer made an error when computing the execution cycles of the instructions involved, and possibly the comment is the only error. The routine may have originally been written for 12-cycle loops during Apple II development, then changed to 11-cycle loops because some marginal tolerance components would not work with 12 cycles. Perhaps they forgot to change the comment. Whatever the reason, this comment in the Apple II and IIe reference manuals indicates 12-cycle loops while the routine utilizes 11-cycle loops.*

The basis of the workings of PREAD is this: you want a number between 0 and 255 returned. The timer duration will vary between 2 and 3302 microseconds with a 150000 ohm paddle and a .022 microfarad input capacitor. However if the values of both of these components are 10% low, the timer duration will vary between 2 and 2673 microseconds. The 256 loops of 11 cycles take 2760 microseconds in the Apple. 256 loops of 12 cycles take 3011 microseconds. Eleven-cycle loops are a good duration to use with plus or minus 10% tolerance components, but a very small number of resistance/capacitance combinations might never allow the PREAD routine to reach a count of 255. It is possible that Apple specifies plus or minus 5% on the capacitors or the potentiometers. Most Apple paddles have a large slack area on the clockwise side where PREAD returns 255 no matter where you set the paddle. This is because PREAD allows for component tolerance. An improved Apple would have a large resistance trimmer pot across each paddle which would let the owner calibrate his paddle set to 11-cycle polling loops.

So the PREAD routine polls the timer in a loop which takes into account realistic possibilities of component variation. That is all to the good, and PREAD is an adequate utility for many purposes. There are, however, some weaknesses in PREAD. Try running the following Applesoft program:

```
10 FOR A = 0 TO 500 : NEXT :
   A = PDL(0)
20 B = PDL(1) :  HOME :
   PRINT A;"---";B :  GO TO 10
```

It simply reads the paddle inputs and prints them. The FOR/NEXT loop is a short delay to minimize screen flicker. With the program running, leave Paddle 1 fully clockwise and change Paddle 0 to some low setting. The result is that Paddle 0 interferes with Paddle 1. This is because all four timers are triggered every time $C07X is accessed. In the Applesoft program, the "A = PDL(0)" causes PREAD to be called with 0 in the X-register. This triggers all four timers, and when the Timer 0 pulse is short, the PREAD routine is exited in a relatively short period of time. However Timer 1 will still be set if Paddle 1 is further clockwise than Paddle 0.

*Another interesting comment error in the monitor listing can be found at $FA6F—$FA74 of Apple II Autostart monitor and Apple IIe monitor listings. The code here brings AN0 and AN1 to TTL low, but the comment and mnemonic labels indicate that AN0 and AN1 are being brought to TTL high.

When the "B = PDL(1)" statement is executed, PREAD is entered with X = 1, and the timers are triggered again. But the timer pulse may still be high from the previous PREAD routine which read Timer 0, and **the timers are not retriggered by C07X' if they have not yet reset from the previous trigger.** This means that the Timer 1 pulse will be dropping after a short period of time after PREAD is entered, even though Paddle 1 is a long ways clockwise.

There are two ways which this sort of interference may be avoided in BASIC programs. One is to always ensure there is a time delay between reading different paddles. It does not take much in BASIC. "15 FOR C = 0 TO 0 : NEXT" in the above program does the trick, or just insert a few instructions between PDL(n) expressions. The second way is to poll the timer in BASIC to make sure it is reset before trying to read it. In the above program: "15 IF PEEK(−16283) > 127 THEN 15". Actually, the delay in this last cure is probably long enough to ensure that Timer 1 is reset by the time −16283 is actually examined, but you will be sure if you use it.

The interference between timers is more pronounced when calling PREAD from assembly language programs. This is because machine language is so fast that hundreds of instructions can be executed after a PREAD routine, and some of the timers may still be set. It is also possible for a PREAD to a timer to interfere with a subsequent PREAD to the same timer. Consider the following program sequence:

```
        LDX  #Ø
        JSR  PREAD
        JSR  PREAD
        STY  SAVEPØ
```

You may have wanted to cause a paddle variable delay with this sequence. Now if Paddle 0 is fully clockwise, the first PREAD is done normally, but the second one returns a low value instead of 255. This is because the first PREAD is exited as soon as 256 polling loops have been performed. Timer 0 is still set though, and it is therefore already set when the second PREAD is entered. The result is a return value in the Y-register of 70 or so instead of the expected 255.

Misreading a timer due to a previous call to PREAD can be avoided by preceding all calls to PREAD with a check like this:

```
        LDX  PDLNUM
NOTRDY  LDA  PDLØ,X
        BMI  NOTRDY
        JSR  PREAD
```

This simply waits until a timer is reset before attempting to trigger and read it. Of course, there would have been no problem in BASIC or assembly language if this check had been included at the beginning of PREAD in the motherboard firmware.

An aspect of PREAD that should be well understood is that it takes a long time, and that the time it takes gets longer as you turn the paddle clockwise. This can be used to advantage in a paddle variable delay routine which allows the user to vary execution speed by adjusting a paddle. More often, the time delay is a nuisance, causing unwanted time delays in a computer that can't afford them. One way to speed programs calling PREAD is to only use counts 0—63. The paddle tweaker becomes aware that there is no control when he goes too far clockwise and controls his Formula-1 racer with less paddle range. Average paddle reading time is reduced by 75% and the sensitivity of the computer action to the paddle tweaker's touch becomes greater. This is tolerable with a paddle set but less tolerable with a 1-inch joystick which is already very sensitive to the touch.

Assembly language programmers should not feel tied to PREAD. PREAD is handy and often adequate, but it's not the last word in reading paddles. Figure 7.9 is a program which reads Paddle 0 and Paddle 1 simultaneously, an obvious capability since all timers are triggered simultaneously. This paired paddle poller polls the pair of paddles precisely in 22-cycle loops, and therefore returns values equal to one half of what they would have been if read by PREAD. The Paddle 0 value is returned in the Y-register and the Paddle 1 value is returned in the X-register. The values can each be shifted left one bit for compatibility with PREAD if this is desirable. The advantage of reading the two paddles together is that paddle reading time is cut in half. The full mechanical range of each paddle is used and the number returned is 0 to approximately 160. It can be argued that no resolution is lost since 1/256 resolution exceeds the practical resolution of a ¾ inch diameter carbon potentiometer and possibly the stability of a 558 timer. In other words, it's hard to find a point on the pot where the PREAD routine returns a single value that does not jump back and forth between readings. It is also very hard to adjust the paddle so as to increase or decrease the returned value by one. Resolution of 1/160 is easily good enough for this hardware.

A final recommendation for speeding paddle reading is to integrate the timer polling with other program execution. The time delay problem exists

```
SOURCE FILE: SIMULREAD
0000:              1 ************************************************************
0000:              2 *                                                          *
0000:              3 *                                                          *
0000:              4 *      SIMULTANEOUS READ OF PADDLE-0 AND PADDLE-1          *
0000:              5 *                                                          *
0000:              6 *                      BY JIM SATHER                       *
0000:              7 *                                                          *
0000:              8 *                         1/24/83                          *
0000:              9 *                                                          *
0000:             10 *                                                          *
0000:             11 ************************************************************
0000:             12 *
0000:             13 *
0000:             14 * PADDLE-0 DIVIDED BY 2 IN Y-REG
0000:             15 * PADDLE-1 DIVIDED BY 2 IN X-REG
0000:             16 *
0000:             17 * SIMULTANEOUS READ PROGRAM LOOP IS 22 CLOCKPULSES.
0000:             18 *
0000:             19 * MONITOR PREAD ROUTINE PROGRAM LOOP IS 11 CLOCKPULSES.
0000:             20 *
0000:             21 *
C064:             22 PDL0     EQU  $C064
C065:             23 PDL1     EQU  $C065
C070:             24 PTRIG    EQU  $C070
0000:             25 *
0000:             26 *
----- NEXT OBJECT FILE NAME IS SIMULREAD.OBJ0
1F00:             27          ORG  $1F00
1F00:AD 70 C0     28 DOIT     LDA  PTRIG
1F03:A2 00        29          LDX  #0
1F05:A0 00        30          LDY  #0
1F07:48           31          PHA              GIVE SOME SPACE FOR COUNT = 0
1F08:68           32          PLA
1F09:24 00        33 GOTPDL1  BIT  $0
1F0B:AD 64 C0     34 CHKPDL0  LDA  PDL0
1F0E:10 0D        35          BPL  GOTPDL0
1F10:EA           36          NOP
1F11:C8           37          INY
1F12:AD 65 C0     38          LDA  PDL1
1F15:30 02        39          BMI  NOGOTS
1F17:10 F0        40          BPL  GOTPDL1
1F19:E8           41 NOGOTS   INX
1F1A:4C 0B 1F     42          JMP  CHKPDL0
1F1D:             43 *
1F1D:             44 *
1F1D:24 00        45 GOTPDL0  BIT  $0
1F1F:AD 65 C0     46          LDA  PDL1
1F22:30 F5        47          BMI  NOGOTS
1F24:60           48          RTS

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

Figure 7.9 Assembler Listing: A Paddle Read Program.

as long as normal program flow must wait thousands of microseconds for a timer to reset. When the paddles must be read often and speed is important, it may be necessary to arrange routines so that they can check the timer states occasionally, only interrupting program flow momentarily. For example, suppose that you are computing HIRES plot coordinates in a 100-cycle loop. At the end of each loop, you can check the previously triggered timer to see if it has reset yet and increment a counter if it hasn't. You wind up reading the timer with a resolution of about 1/33 which really is sufficient for many tasks. Of course, this would be a complicated program, but the results would be rewarding. Complicated programs are within the capabilities of any reader of this book who has the time and the urge.

HARDWARE APPLICATION

EXTENDING THE GAME I/O SOCKET

Have you ever seen an Apple with two joysticks plugged in? Why not? It's a capability of the Apple. The answer is that when a standard joystick or paddle set is plugged in to the game I/O socket, the pins for one pushbutton input, two paddle inputs, four annunciator outputs, and the C040 STROBE' become inaccessible there. It would do you no good to plug another joystick into the extension jack in the back, because all the standard joysticks support only Paddle 0 and Paddle 1. Your two joysticks would just interfere with each other.

Now I don't mean to imply that the extension jack gives no added capabilities. To the contrary, when a joystick or paddle set is installed in the extension jack, all of the game I/O signal lines are still accessible at the game I/O socket. Additionally, even when a plug is installed in the extension jack, it is fairly easy to attach a spring loaded clip to any of the signal lines on the back of the extension jack. But the fact remains, if you want to switch between paddle and joystick or use two joysticks connected simultaneously to the four Apple timers, you need to use some sort of extension device which supports the capability.

Several game I/O extenders are commercially available for the Apple. This application note shows two extension circuits you can build yourself. One is simple, allowing you to plug a joystick or paddle set into the game I/O socket and still have a 16-pin DIP socket available with the remaining I/O pins accessible. The other is more complex, allowing you to have two paddle sets and two joysticks simultaneously connected with switched control between paddles or joysticks. This **game I/O extender** also contains an extension socket for connection to other devices.

Let's look at the simpler circuit first, pictured in the photos of Figure 7.10. This is a paddle set with an extension socket soldered on top of its 16-pin plug. Pins 6 and 10 are removed from the upper socket because these are the PDL 0 and PDL 1 inputs which are being used by the paddle set. PB0 and PB1 are fed to the extension socket even though they are used by the paddle sets. Switch inputs can be paralleled, so one of several switches can operate a given pushbutton input. Potentiometers, however, cannot be connected simultaneously to a timer input. They would interfere with each other.

The benefit of the extension socket is that with the extended paddle set installed in the game I/O socket, the other signal lines are still accessible there. But the modification must be performed carefully to ensure mechanical strength. The first step is to buy a high quality 16-pin DIP socket. You will also need to buy a 16-pin plug and cover like the one used on Apple paddle sets designed for the game I/O socket. We assume that the plug to be modified on the paddle set or joystick is so thoroughly glued and sealed that you cannot hope to solder a socket to it. Here is the procedure to mount the extension socket on your paddle set:

1. Separate the cover from the plug on your paddle set. If you think you can solder a socket to this mess, proceed to step 8.
2. If you cannot salvage the old plug, remove the two resistors from it (if they are there), and cut the wires from it which lead to the paddles.
3. Strip one inch of the outer insulation from the wire bundle going to each paddle. This exposes three insulated wires in each bundle. If Apple is consistent, the green wire goes to +5V, the white

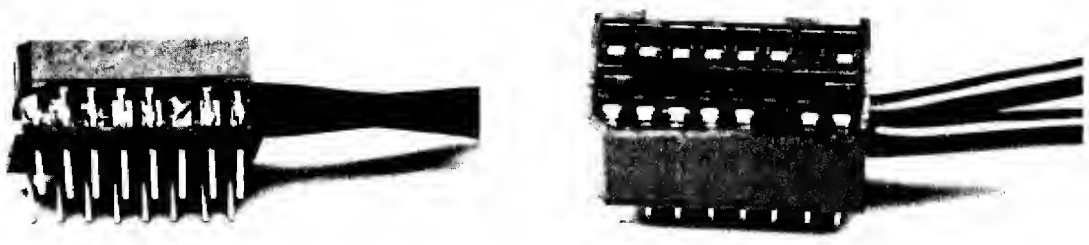


Figure 7.10 A Modified Game I/O Plug.

wire is the pushbutton wire, and the black wire is the paddle wire. This should be verified with an ohmmeter. With the ohmmeter, find the two wires connected to the pot. The resistance across them will vary between 0 and 150,000 ohms as the paddle is turned. The wire left over is the pushbutton wire. Now find which wire is shorted to the pushbutton wire when the pushbutton is pressed. This is the 5 Volt wire, and the other wire is the paddle wire.

4. Some paddle sets have a fourth wire going from the plug to the paddles. This wire is ground and should be connected to pin 8 of the plug. Presence of the ground wire idicates that pushbutton pull-down resistors are mounted in the paddles instead of the plug. The ground wire can be identified because there will be 200—1000 ohms resistance between it and the pushbutton wire.
5. Cut the paddle wires for each paddle back ½ inch. Leave the other four wires at their present length. Strip ⅛ inch of insulation off the end of all six wires.
6. Figure 7.11 shows the wiring of the plug. Install the two resistors first (if they were there), making sure the leads do not extend very far beyond the solder posts. If you are certain that the paddle set will be used only in an Apple IIe and never in an Apple II, do not install the resistors. If there are no resistors and you wish to use the paddle set in an Apple II, install 560 ohm resistors as shown in Figure 7.11. Make certain the resistors are not in the paddle body before doing this (see step 4).
7. Connect the wires and solder. Use a low wattage iron and do not overheat the pins or the plastic base will be damaged. Insert the plug into a spare socket while soldering to keep the pins aligned if the plastic becomes soft from overheating.
8. If necessary, sand down the corner of your socket so the plug cover will slip over it. Pull pins 6 and 10 out of the socket or cut them off if the plastic is molded around the pins.
9. Fit the socket over the paddle set plug and hold this assembly lightly together in a soft jawed

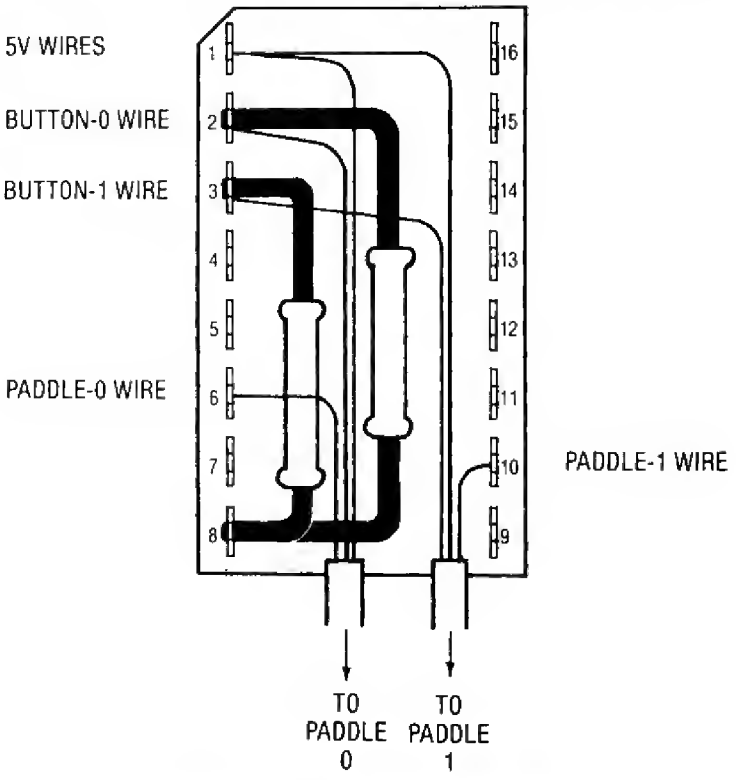


Figure 7.11 Wiring a Paddle Set Plug.

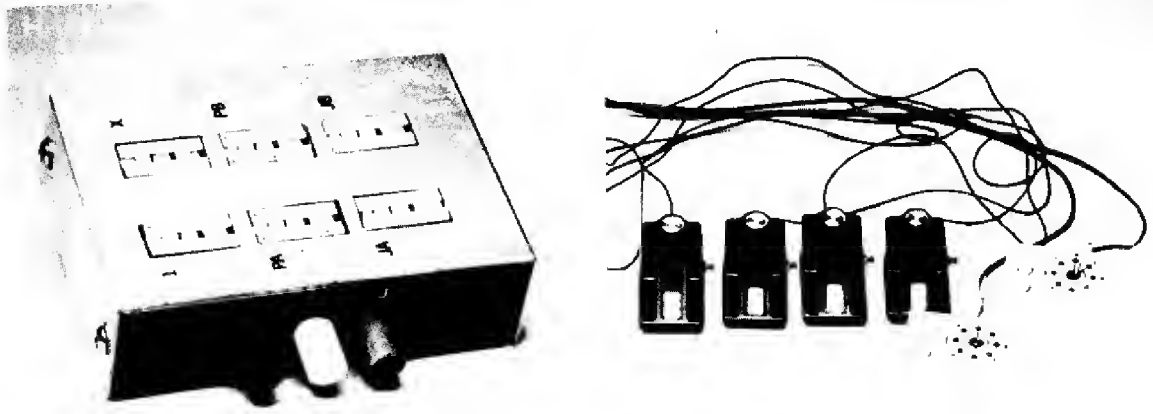


Figure 7.12 This Game I/O Extender Can Support Two Sets of Paddles and Two Joysticks Simultaneously.

vise. All wires should be dressed inside the pins of the socket and the socket pins should be outside of the plug pins. Solder the 14 pins of the socket to the appropriate pins on the plug.

10. Check out the operation of your paddle set and extension socket.
11. If you desire, fill the area between the plug and expansion socket with epoxy or a sealant like RTV. This will give your assembly more mechanical strength. Do not seal the assembly until you are certain it works correctly.
12. Cut the top off the plug cover so the topless plug cover is 5/16 inches high. Cut out a small notch for the wires to pass through. Slip the cover over your assembly and glue it on with a small amount of epoxy cement. Remember that you may want to get back in there some day.

Figures 7.12 and 7.14 are photos and a schematic diagram of the more ambitious **game I/O extender**. This unit is meant to sit outside of the computer case, connected to the game I/O socket via a 16-pin DIP jumper. It is basically six 16-pin sockets wired together with some configuration switches. The scheme is this: two of the sockets are meant for paddle sets. One of the paddle sockets is connected normally but the other is connected so a standard paddle set will control Timers 2 and 3. Two of the sockets are meant for joysticks. The joystick sockets are wired so all four timers are utilized by two joysticks. Switch S2 places the joysticks in one of two possible configurations as shown in Figure 7.13. The paddles and joysticks may be connected at the same time. Switch S1 enables either the joystick or the paddles.

A third switch is necessary if you wish to use the extender with an Apple II with the SHIFT key mod installed. The SHIFT key mod works by connecting the SHIFT key to PB2, but neither the SHIFT key mod nor a pulled down pushbutton will work if both are connected to PB2 at the same time. The game I/O extender allows you to have both installed by selecting between them via S3. The normal SHIFT key mod is to connect one of the SHIFT keys to pin 4 of the game I/O socket. With the game I/O extender, the SHIFT key should be connected to pin 16 (normally not connected) of the game I/O socket. Then switch S3 can select between pin 16 and a paddle or joystick pushbutton for routing to the PB2 input.

In the Apple IIe, the shift key mod is available via the X6 motherboard jumper. When this jumper is made, a pulled down pushbutton cannot be connected to PB2 for reasons cited in the previous paragraph. If you must have the SHIFT key mod in the Apple IIe, two alternatives are: leave S3 in the SHIFT key mod position when the game extender is installed in an Apple IIe with X6 soldered, or connect a wire from the SHIFT' line to pin 16 of the game I/O socket instead of soldering X6.

The extender construction technique is to mount the six sockets on a general purpose IC board. Use the type with feed through solder holes so wires can be connected on both sides of the board. The board is mounted in a case with six holes through which the sockets fit. A nibbling tool is good for cutting the holes. The installation procedure is: install the sockets in the board; wire the board and switches as shown in Figure 7.14; mount the board and switches in the case. The appearance of your extender will vary with your selection of switch styles and enclosure. Enjoy your extender. It's really pretty handy.

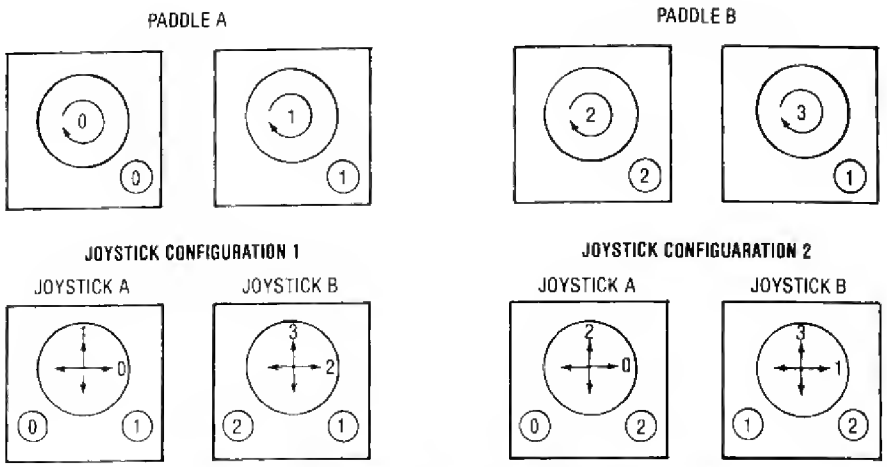


Figure 7.13 Game I/O Extender Configurations.

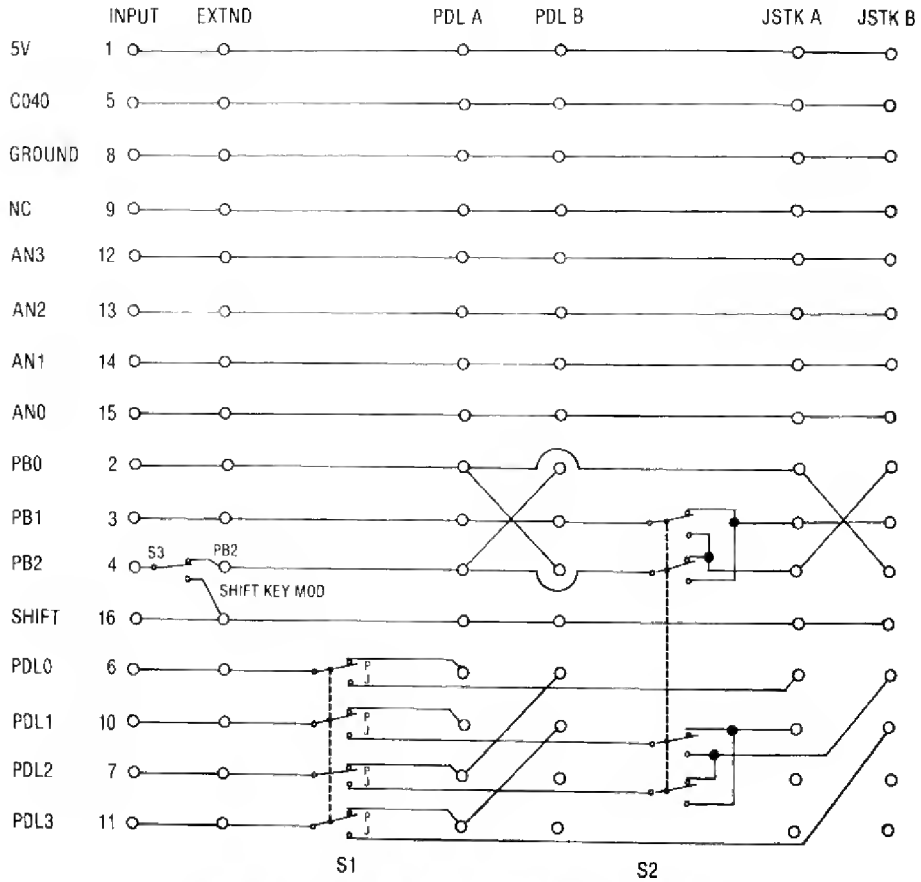


Figure 7.14 Schematic: A Game I/O Extender.

HARDWARE APPLICATION

GAINING ACCESS TO THE ALTERNATE KEYBOARD SET

The keyboard circuitry of the Apple IIe is typically Apple. Man, this circuitry is neat, a versatile design which makes the Apple IIe superior to other computers. Also typical of Apple, the keyboard versatility is not advertised as a selling point, it is not documented for people who don't read schematics, and minor changes are made which make it difficult for information disseminators like myself to devise clear, concise descriptions. Hey up there at Apple. We're not witless walking wallets down here. We can appreciate the finer features of the Apple IIe.

Enough criticism. The Apple IIe does have a very versatile keyboard circuit which enables access to an alternate keyboard layout and easy redefinition of the keyboard and numeric keypad layouts. Also, the inclusion of Dvorak as an alternate keyboard layout might lead to similar action by other manufacturers and eventually lead to acceptance of Dvorak as the English language standard keyboard layout. Conventional wisdom is that Dvorak doesn't have a chance because too many people have learned QWERTY, and normally available typing machines are all QWERTY. But suppose Dvorak becomes available as an alternate layout on virtually all typing machines. Wouldn't some people convert to Dvorak, and wouldn't typing instruction courses begin to teach Dvorak? If the seedling takes root, we all may be touch typing at a higher pitch in a generation or two.

There are other reasons for accessing the alternate keyboard set than learning to touch type in Dvorak. Suppose you have a numeric keypad and wish to assign special function ASCII to the various keys so your application software is easier to operate. For example, if you use *Applewriter* a lot, you could assign the ASCII of the various *Applewriter* control keys to the alternate numeric keypad junction code. Normally your keypad would have numeric ASCII, but selecting the alternate set would make your keypad an *Applewriter* controller. Of course, you would need to program a custom keyboard EPROM with your desired alternate keypad layout to gain this feature.

Other uses for the alternate set are possible. The question is, "how do you access the alternate set?" The answer is "there are several ways, none of which is particularly complex." Read on.

If you have a Revision A motherboard, you can access the alternate set by soldering the X2 jumper and cutting the X1 jumper. Then AN2 will switch between keyboard layouts with the alternate layout selected when AN2 is set ($C05D). You can then switch between layouts at anytime from a program, or at the keyboard by accessing $C05C/D or −16292/1.

You can also make AN2 switch between keyboard sets with Revision B motherboards, but it is more involved because the keyboard ROM A10 input is connected to ENVID'. You can make AN2 control ENVID' by soldering the X3 jumper, but enabling the alternate keyboard layout will disable video generation. I don't know what Apple was thinking of when they tied ENVID' to the keyboard ROM.

If you really want AN2 control of the keyboard layout on a Revision B motherboard, perform the following:

1. Cut the X2 jumper to isolate ENVID' from the keyboard ROM.
2. Solder a short wire between the back half of the X2 jumper and the back half of the X3 jumper (don't solder the two halves of X3 together). The back half is the half situated toward the back of the motherboard.

A problem with using AN2 to switch between keyboard layouts is that the alternate set will always be selected after a system reset. It is true that all the annunciators are reset when RESET' drops low, but the firmware RESET' handler proceeds to set AN2 and AN3 (I don't know why). The result is that your keyboard will be set to Dvorak at power up and whenever RESET is pressed. This is probably undesirable, but you can reverse the situation by installing a keyboard EPROM identical to the normal keyboard ROM but with the top half swapped with the bottom half. Then the QWERTY layout will be selected by system resets.

I recommend installing a manual switch to select between keyboard layouts rather than depending on AN2. The reason is that you never know what some program is going to do with AN2, and you might find yourself in Dvorak when you want to be in QWERTY. Other reasons are that a special keyboard EPROM is not required, and that this is the sort of function over which manual control is best.

The manual switch needs to bring pin 19 of the keyboard ROM high or low (see Figure 7.4). Here is one installation that will work.

1. Solder wires to the bases of pins 12, 19, and 24 on a 24-pin IC socket.
2. Solder the other ends of the wires to a single pole, double throw switch such that pin 19 is connected to the common terminal, and pins 12 and 24 are connected to the other two terminals.
3. Remove the keyboard ROM and install the socket/switch assembly in the vacated motherboard socket.
4. Install the keyboard ROM in the socket/switch assembly.
5. Mount the switch in a convenient hole at the back of the Apple IIe.
6. If Rev A, cut jumper X1 to isolate pin 19 of the keyboard ROM from ground.
7. If Rev B, cut jumper X2 to isolate pin 19 of the keyboard ROM from ENVID'.

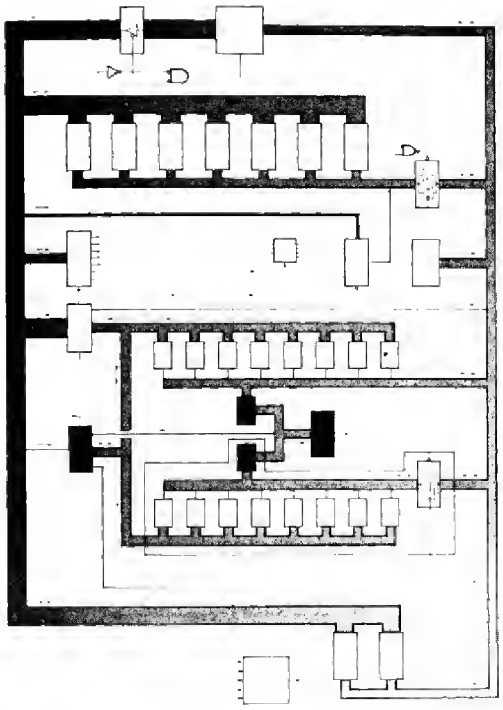
The switch will now place either 5 volts or ground on pin 19 of the keyboard ROM, thus selecting between character sets.

Are two keyboard layouts enough for you? If you need more, you can carry the above installation procedure a step further to mechanize switching between four sets on a 4K EPROM (2732). The 2732 is compatible with the 2716, except that pin 21 is A11 instead of VPP. Perform the following to adapt to 2732:

1. Wire a 24-pin IC socket as instructed in the above installation procedure.
2. Bend pin 21 of the wired socket out so it will not make contact when the socket is inserted into the keyboard ROM socket on the motherboard.
3. Solder a wire between pin 21 of the socket and the common terminal of a second single pole, double throw switch.
4. Solder a short jumper between the terminal of the first switch that is connected to pin 24 of the socket and one of the two terminals on the second switch.
5. Solder a short jumper between the terminal of the first switch that is connected to pin 12 of the socket and the remaining unconnected terminal of the second switch.
6. Install the socket/switch assembly as above, mounting the two switches near each other on the back of the Apple IIe. The 2732 EPROM with your four keyboard layouts is installed in the wired socket which, in turn, is installed in the keyboard ROM socket of the motherboard.

The preceding paragraphs mentioned several instances in which you might wish to program your own keyboard EPROM. There are other reasons for doing this. One reason is to operate with a numeric keypad that wasn't originally designed for the Apple IIe. This is done simply by reprogramming that part of the keyboard ROM assigned to the numeric keypad. Another reason is to make a minor change in the standard layout to suit your preference. For example, I use both a Kaypro and an Apple IIe, and am annoyed by the fact that the arrow keys on the Apple are laid out "left, right, down, up," and the arrow keys on the Kaypro are laid out "up, down, left, right." I don't care how they lay out the arrows as long as they're the same so I can get used to them. The solution is to reprogram the keyboard ROM so that the Apple IIe keys are laid out like the Kaypro keys. Then just pull off the key caps and reinstall them in the new order.

Table 7.2 provides a useful guide for persons wishing to program their own keyboard EPROM for any reason. EPROM addresses for any key can be computed from the Table 7.2 base address using the guide at the bottom of the table. Use of 2732 will create a second table, identical to Table 7.2 except with base addresses increased by $800.



chapter 8

Video Generation

The marriage of data processing and video display technology has been one of the most important developments in the advance of computers. Combined with the keyboard, the video display provides a direct communication link between people and computers that makes the old, expensive, physically large computers seem to be just machines. Imagine communication with your Apple using a teletype terminal with no video display. How would that affect important applications like word processing, spread sheet accounting, data base management, graphics display, and Donkey Kong? Without video, the Apple wouldn't be worth owning.

Of course, the Apple does have a video display capability, and a large portion of the motherboard hardware is related to the generation of video. We have seen in other chapters that many features of bus structure, timing, and RAM addressing in the Apple IIe are dictated by the fact that the dissimilar tasks of stored program execution and video display generation are performed simultaneously in this computer. Additionally, the **video scanner** (internal to the IOU) and **video generator** (partly internal and partly external to the IOU) are functional areas that exist for the sole purpose of making up the video display. All of these functional areas are interconnected in a scheme which allows Apple programs to control the video output.

Figure 8.1 is a simplified diagram of the video display control processes of the Apple IIe. As Figure 8.1 shows, the MPU controls the output of video in a very indirect way. Under direction of the controlling program, the MPU sets the screen mode, computes a correct address in memory, and stores selected code at that memory address. In so doing, the MPU is setting up a small area of the **screen map**. The extent of MPU involvement and, by extension, programmer involvement in outputting video consists entirely of setting up this screen map. The actual output of video is controlled by the video scanner which scans memory and drives out the map, and by the video generator which processes the map to produce the **VIDEO signal**. You can actually stop the MPU by pulling READY or DMA' low, and the Apple will continue to output to the screen the map which was set up by the MPU before you stopped it.

Compare this indirect MPU involvement to a printer output port where the MPU—under program control as always—actually stores coded data

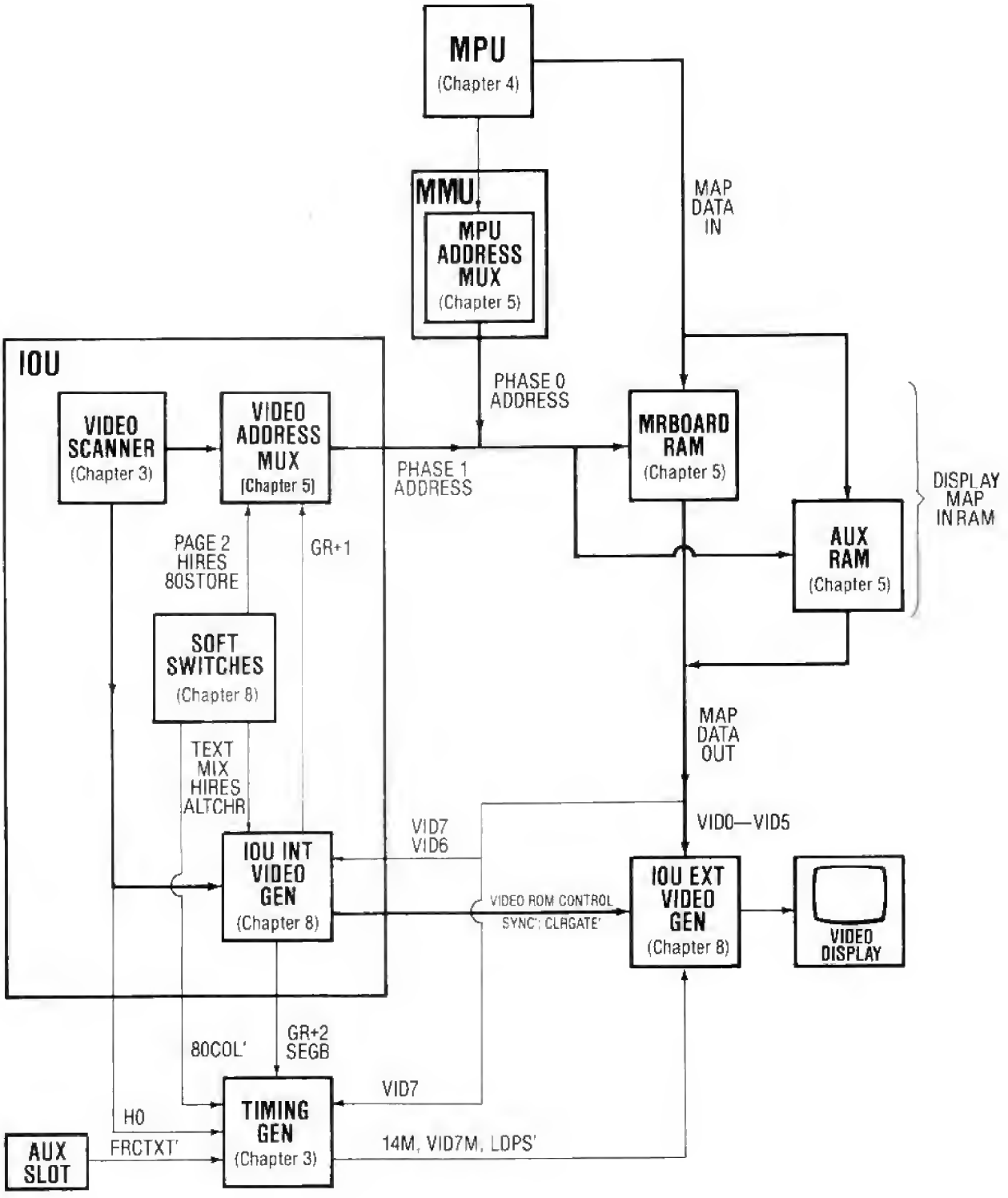


Figure 8.1 Overall Video Generation.

at a special address to output it to the printer. The sneaking access to RAM by the video scanner is an example of **DMA**, which is like someone else sleeping with your spouse. RAM in the Apple is very promiscuous. It goes to bed with the video scanner every other night. Then on most off nights, it goes to bed with the MPU. The MPU has no idea what unfaithful RAM is up to during PHASE 1.

The video generator must take the offspring of the scanner/RAM affair and interpret it as text, LORES graphics, or HIRES graphics to produce a signal which causes a television or monitor to produce the computer display. This signal is referred to as the VIDEO signal, and it is one of the more complex signals in the Apple. The purpose of this chapter is to discuss the nature of the VIDEO signal, how it is produced in the video generator, and operational features of the Apple IIe resulting from the way the VIDEO signal is produced. Other chapters of the book contain detailed descriptions which are important in achieving a broad understanding of how the tasks of video generation and program execution are integrated in the Apple IIe. These include descriptions of overall video generation (Chapter 1), video scanning within the context of bus structure (Chapter 2), the video scanner (Chapter 3), and RAM addressing (Chapter 5). The subject at hand is the video generator, and we begin our discussion with a description of the Apple IIe VIDEO signal.

THE APPLE IIe VIDEO OUTPUT SIGNAL

Let's watch a television show for a minute; how about . . . *Taxi*? That Louie is really something. The picture we see originates with a camera which outputs **composite video**, a signal composed of picture, synchronization, and color information. This signal is routed to a transmitter which modulates a **radio frequency** signal with the composite video and with audio from a microphone. The radio frequency signal is distributed nationwide to local stations which transmit the signal to receivers in their areas. The television in your home is a receiver/processor which extracts the audio and composite video from the radio frequency signal and processes it to form the picture we see and the sound we hear.

The previous paragraph could be describing television in any number of countries or continents in the world which broadcast television signals based on similar principles. The exact details of various signals vary, however, among several standard systems used in various areas of the world. The American standards were formulated by the **NTSC** (National Television System Committee) and adopted by the FCC, which allowed black and white television broadcasting after July 1, 1941. Updated NTSC standards for color television broadcasting were adopted by the FCC on December 17, 1953. The American television must be designed to receive and process NTSC standard signals.

When the original Apple II was built, the FCC frowned on the idea of computers outputting radio frequency signals to a television because it is possible for a tiny amount of the computer signal to be radiated and cause interference with television reception in the neighborhood. Please note that this level of radiation leakage is not a health hazard and is smaller than the man-made electromagnetic fields in which we all live. To avoid conflict with FCC regulations, the Apple II was designed to output composite video which will drive an NTSC standard composite video monitor. Later, the Apple IIe was designed to output video which is compatible, and in most respects identical, to the video which is output from the Apple II.

If you modulate a radio frequency signal with the Apple's VIDEO signal, that radio frequency signal will drive an NTSC standard television receiver. Of course, your thirty dollar RF modulator may tend to leak RF radiation and interfere with neighborhood television reception. The television takes the radio frequency signal and converts it back to the same VIDEO signal that left the Apple's video output jack. From this point in its circuitry, the television is identical to a composite video monitor.

Figure 8.2 shows the characteristics of the Apple VIDEO signal. It is made up of three components: the **PICTURE signal**, **SYNC**, and the **COLOR REFERENCE BURST**. The signals are added together in such a way that a television can tell them apart. The television can separate the SYNC from the VIDEO signal because, during SYNC pulses, the VIDEO signal is at a lower voltage than at any other time. It also can detect the COLOR BURST, because it knows where to look for it—right behind the horizontal sync pulse on the "back porch" of the horizontal blanking gate.

There is a voltage point on the VIDEO signal called the **black reference**. Voltages above the black reference cause the picture tube electron beam to strike the interior face of the tube with enough velocity for light emission to result. The VIDEO signal goes above the black reference only when it is time to paint, and it stays below the black reference the rest of the time. The SYNC pulses are blacker than black, extending below the blanking signal to a point where they are detected as sync, not a picture signal, by the television. We thus have

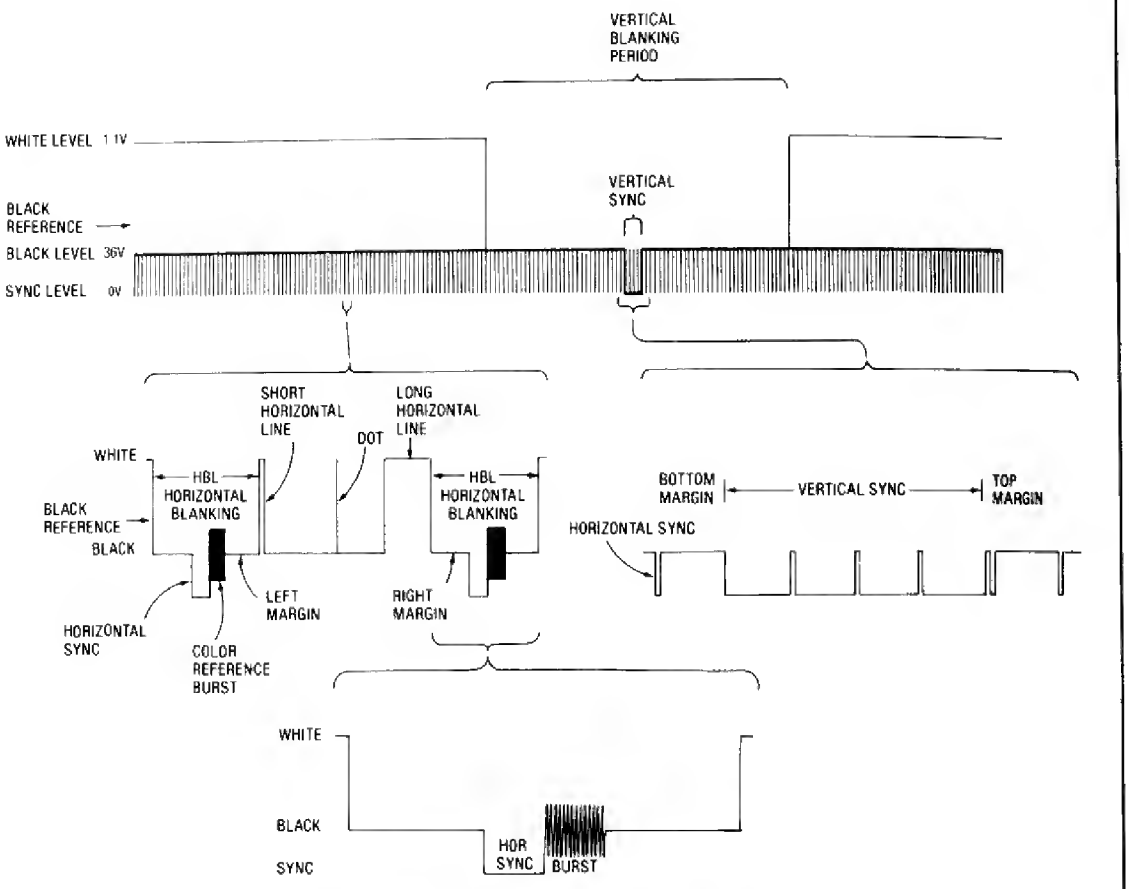


Figure 8.2 The Apple IIe VIDEO Signal.

three signal levels, the white level, the black level, and the sync level. The Apple signal is immensely less complicated than normal television composite video in this regard. The level of the picture signal in composite video can be any voltage between the black level and white level at any instant. This is the way television reproduces the remarkable variety of lighting shades found in a normal television picture. Even black and white television is really black and white and innumerable shades of gray. The Apple video, when color information is not present, is truly black and white.

The horizontal and vertical sync pulses both descend below the black level. Any sharp negative edge in the sync level is interpreted by the television as horizontal sync. Any long duration negative pulse in the sync level is interpreted as vertical sync. The vertical sync pulse in the Apple lasts for four complete horizontal scans. As in normal television video, there are sharp serrations in the vertical sync pulse so that horizontal sync can be detected, even in the middle of the vertical sync pulse.

There are 262 horizontal sync pulses for every vertical sync pulse in the American Apple. The 4-cycle horizontal sync pulse occurs in the middle of **HBL**, the 25-cycle **horizontal blanking gate**. During HBL, the VIDEO signal is held below the black reference level, creating the right and left black margins on the screen (see Figure 8.3). The picture signal occurs between the horizontal blanking gates, naturally, creating the screen display between the left and right margin. The vertical sync pulse occurs in the middle of **VBL**, the 4550-cycle (70 horizontal scans) **vertical blanking gate**. The VIDEO signal

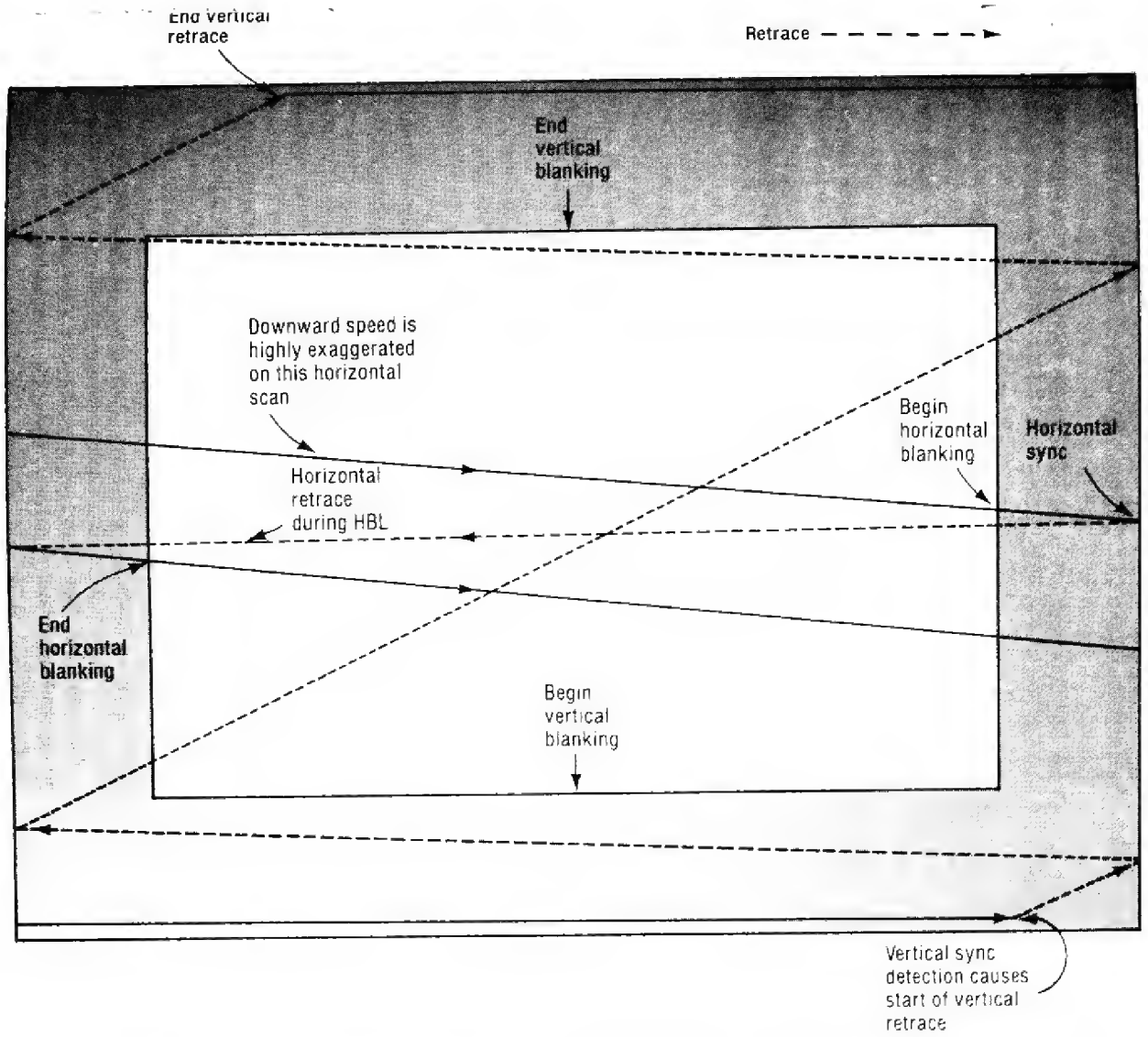


Figure 8.3 Blanking During Television Scanning Causes the Black Margin Around the Apple Display.

is held below the black reference level during the entire VBL period, creating the lower and upper black margins on the screen.

Assume the electron beam is at the top left corner of the screen, and that the Apple display window on your television is 10 inches across. The beam is moving left to right as you look at the screen, at about 10 inches per 40 microseconds or about 14,000 miles per hour. Since we are at the top, the vertical sync pulse has just occurred and it is the second half of VBL. The beam scans across to the right side of the screen, but we don't see light on the screen because VBL holds the VIDEO signal in the black. The horizontal sync pulse causes the beam to retrace very rapidly to the left side (none of this slow poke 14,000 mph stuff) so it can scan across left-to-right again. This cycle continues as the beam moves across again and again and less speedily down the screen to the first displayed line, about an inch from the top of the screen.

In the last undisplayed line, VBL ends about an inch from the right side, but HBL begins at the same time, so the screen is still blanked. After the beam scans past the right edge, horizontal sync occurs, causing retrace, and the beam begins the first displayed line. When the beam gets about an inch from the left side of the screen, HBL ends and the VIDEO signal begins switching back and forth between the black level and the white level, decreasing and increasing the energy in the electron beam to cause bright spots and lines on the screen interspersed with black spots and lines.

About an inch from the right side of the screen, HBL forces the VIDEO signal into the black where it remains until the beam has scanned past the left margin of the next line. This cycle continues for 192 displayed horizontal scans. At the end of the display period of the last displayed line, VBL and HBL begin together, marking the start of the bottom margin. The beam scans the rest of the way to the bottom in the black, then the vertical sync pulse causes a rapid retrace to the top of the screen, where we began our description of the continuous cycle of the electron beam.

In NTSC standard television scanning, a process known as **interlacing** takes place. In interlacing, alternate vertical scans are displaced vertically by half the distance between two horizontal scans. This means it takes two vertical scans to actually paint the complete television picture, and it is a tricky way of increasing vertical resolution without increasing flicker. In NTSC scanning, there are 262.5 horizontal scans in each vertical scan for a picture composed effectively of 525 lines. The Apple SYNC is not set up to cause interlacing. It causes a non-interlaced vertical scan of 262 lines, 192 of which are displayed.

COLOR SIGNALS

The COLOR REFERENCE BURST is a 14-cycle sample of the COLOR REFERENCE signal from the timing generator. Color television sets are designed to look for a 3.58 MHz signal after the horizontal sync pulse. From this short burst, the television is capable of reconstructing the whole COLOR REFERENCE signal. It does this by "phase locking" an oscillator to the COLOR BURST. By this method, the COLOR REFERENCE is transmitted to the television, but it is not present at the same time as picture information, so it does not interfere with the picture. Since the COLOR BURST occurs during HBL, it is not displayed on the screen.

The COLOR BURST does not occur if the TEXT soft switch is set. When no COLOR BURST is present, the color generation in the television is inhibited, so elimination of the BURST prevents undesirable coloration of screen text. The COLOR BURST is not inhibited during TEXT time in MIXED mode, so the four lines of text at the bottom of the screen in MIXED displays have coloration (green, violet, and white assuming SINGLE-RES display mode).

Picture information in an NTSC standard television signal is divided into two components, the color component and the brightness component. The **chrominance** signal contains the color information of the picture, and the **luminance** signal contains the brightness information of the picture. The two signals are transmitted simultaneously and processed together in the radio frequency and intermediate frequency stages of a television set. Once the television signal has been converted from radio frequency to composite video, the television separates the chrominance signal from the luminance signal and processes them individually.

The chrominance signal is a highly complex combination of two 3.58 MHz signals 90 degrees out of phase with each other. In an amazing mathematical/electronic manipulation, the chrominance signal contains the color information of each spot on the screen while the luminance signal contains the brightness information of each spot. Part of this mathematical manipulation is that the chrominance and luminance signals are present together in overlapping frequencies, yet do not interfere with each other. In television video processing, the chrominance signal is separated from the composite video,

then red, blue, and green color signals are extracted from the chrominance signal by comparing it to the COLOR REFERENCE.

In Apple video, the PICTURE signal takes the place of the luminance and chrominance signals of normal broadcast NTSC composite video. The PICTURE signal is a simple binary signal that goes high or low in accordance with text or graphics patterns produced from the screen map during display periods.

In television processing, the Apple VIDEO signal is recovered from the modulated carrier wave and is present at the output of the "second detector." The higher frequency components will, however, look different than they did at the video output jack of the Apple. This is because the square waves of the VIDEO signal are converted to their sine wave components that are within the bandwidth of the television signal paths. In televisions with the normal 4.1 MHz IF bandwidth, those square waves greater than about 1.37 MHz are converted to sine waves of the square wave frequency. This includes the COLOR REFERENCE BURST and higher frequency PICTURE signals, such as those which produce color in the Apple's display. This modified Apple video is present at the input to the luminance and chrominance amplifiers, as well as other sections of the television.

The modified PICTURE signal (a high frequency sine wave, a low frequency square wave, or medium frequency combination) is passed by the luminance amplifier and ultimately controls the brightness of the display. If the modified picture signal is oscillating at 3.58 MHz, it will also be passed by the chrominance amplifier to the synchronous demodulator where it is compared with the reconstructed COLOR REFERNCE to produce red, green, and blue color signals. Thus, the Apple VIDEO signals which produce colored displays on the screen are those with a 3.58 MHz PICTURE signal.*

Processing in a composite video monitor is similar to that in the video sections of a television, but the high frequency square waves may or may not have been converted to sine waves by the time they reach the chrominance and luminance amplifier inputs. If they are not already sine waves, the high frequency square waves will be converted to sine waves in the luminance and chrominance amplifiers. In high frequency response monochrome monitors, there is no chrominance amplifier. The video amplifiers of these monitors will pass the square waves of the Apple VIDEO signal with little distortion. An exception is LORES or HIRES80 gray PICTURE signals, which will be converted to sine waves by monitors of less than 21 MHz frequency response.

The features of Apple graphics are largely dependent on the way the Apple passes color intelligence to the television. There is no need to store HIRES color information in memory. Dot position determines color. Think of the savings in memory over a system where the color information of dots is stored as separate intelligence. The flip side of this coin is: think of the elaborate programs required to produce colored displays dependent on dot position.

There are four classes of Apple video concerning color. First is **black**, the absence of luminance. Second is **white**, the absence of color caused by picture signals less than 3.58 MHz. This occurs when two adjacent HIRES40 dots or four adjacent HIRES80 dots are turned on, increasing signal pulse width and decreasing frequency so there is no 3.58 MHz signal for the TV to interpret as chrominance. Additionally, white occurs in a "COLOR = 15" LORES block, which is identical to seven adjacent HIRES40 dots in four adjacent horizontal scans. White also occurs in TEXT mode where there is no COLOR BURST.

Third is **gray**, the absence of color caused by 7 MHz picture signals. This occurs in "COLOR = 5" and "COLOR = 10" LORES blocks and in HIRES80 displays in which every other dot is turned on. LORES colors 5 and 10 are identical to each other in shade and intensity and are the same as white except less bright. White horizontal lines are caused by long periods of white level VIDEO signal. Gray is caused by 7 MHz oscillation of the VIDEO signal between white level and black level. The alternating black level causes gray to be less bright than white.

Fourth is **colored video**. This video is 3.58 MHz whether it is HIRES or LORES. There are four such HIRES40 colors and twelve such LORES and HIRES80 colors. Four of the LORES/HIRES80 colors are identical to HIRES40 colors: green, violet, orange, and blue. The remaining eight colors come in four tones: light and dark blue, light and dark magenta, light and dark blue-green, and light and dark brown. Of these eight colors, six can be produced in HIRES40 as interference between adjacent bytes of graphics data with bit 7 set on one byte and reset on the other.

This explanation of Apple color is based on analysis of the hardware and timing of the video generator. The details are a little involved, but tread thou in my footsteps, tenacious reader, and thy tootsies will not freeze.

*Some exceptions to this rule and further discussion on television processing are contained in a technical note at the end of this chapter.

DISPLAY MAP MEMORY REPRESENTATIONS

The way in which displayed objects are represented in memory varies with the display mode. TEXT characters are represented by ASCII with some variations in the code for NORMAL, INVERSE, or FLASHING characters. GRAPHICS patterns are represented directly in memory with illuminated portions represented by ONE and blanked portions represented by ZERO. Details of these representations are illustrated in Figure 8.4.

TEXT ASCII from the display map is interpreted in one of two ways, depending on the ALTCHRSET soft switch. With ALTCHRSET reset, $00—$3F represent 64 INVERSE upper/special/numeric characters, $40—$7F represent 64 FLASHING upper/special/numeric characters, and $80—$FF represent a complete 128-character NORMAL ASCII set. With ALTCHRSET set, $00—$7F represent a complete 128-character INVERSE ASCII set, and $80—$FF represent a complete 128-character NORMAL ASCII set.

The ALTCHRSET variation affects only the interpretation of INVERSE and FLASHING code, not the NORMAL code. When NORMAL or FLASHING text is to be displayed, the controlling program must coordinate ALTCHRSET configuration with masking of bits 6 and 7 of stored character code to achieve the desired display. A complete list of the TEXT character representations is given in Table 8.4.

The name of the ALTCHRSET soft switch leads one to believe that it lets the user select some alternate to the standard dot patterns which make up Apple IIe TEXT mode characters. This is not true. ALTCHRSET only lets you switch between the funky INVERSE/FLASHING representations of the original Apple II and the businesslike full ASCII INVERSE representation. If you want an alternate character set, you can get it by replacing the INVERSE patterns in the video ROM with your desired set in a custom video EPROM.

In TEXT mode, each 40-byte display line in RAM is scanned eight consecutive times. For example, if $C1 (code for NORMAL "A") is stored at $400, $C1 is driven out of RAM during each of the first eight horizontal television scans just before the beam gets to the first displayed character position. Video generation circuitry interprets VA, VB, and VC from the video scanner to determine which of the eight 7-dot patterns that make up "A" on the screen it is time to shift out. As a result, 40 bytes contain the display intelligence for eight horizontal scans.

Memory scanning in LORES is identical to that of TEXT, with each 40-byte display area scanned eight times consecutively. But, rather than ASCII for a text character, actual dot patterns are stored to represent LORES blocks. Each memory location in the display map contains two 4-dot patterns, the upper block pattern in bits 0—3 and the lower block pattern in bits 4—7. During the first four times a 40-byte display area is scanned (as identified by VC'), the upper block pattern is processed by the video generator, and the lower block pattern is processed during the second four times a 40-byte display area is scanned.

Video generator processing of a 4-dot LORES pattern consists of shifting the pattern to the PICTURE signal at a 14 MHz rate (twice the speed of

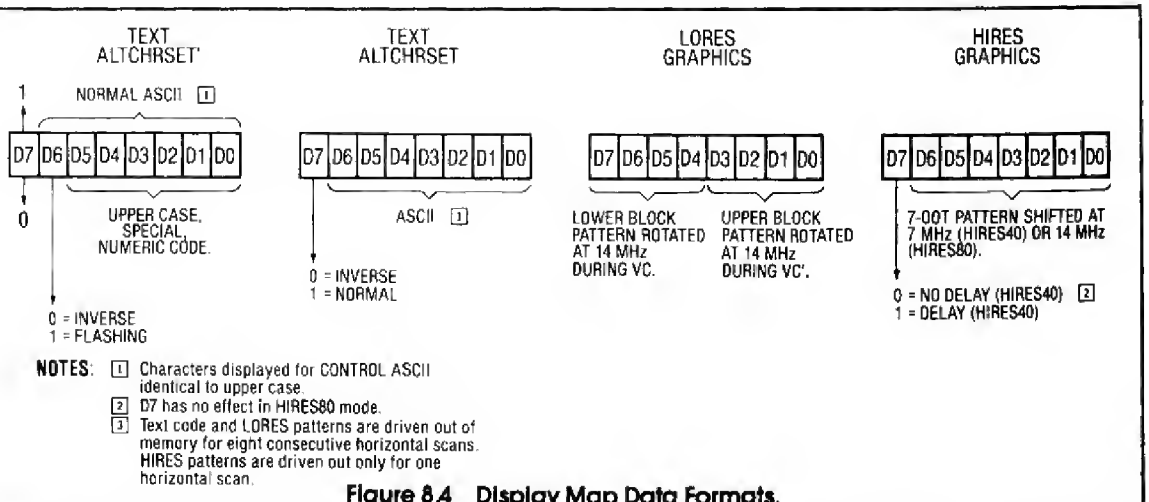


Figure 8.4 Display Map Data Formats.

HIRES40 and TEXT40 patterns). The signal alternations resulting from this high speed shift are too fast to be displayed clearly on a color television, so they are blurred into colored blocks. You can actually see the LORES pattern alternations if you use a high frequency response video monitor with the Apple instead of a television.

HIRES patterns are stored directly in bit 0—6 of display memory locations as 7-dot patterns. Processing of these patterns consists of shifting them to the PICTURE signal, seven dot positions per microsecond (HIRES40) or seven dot positions per ½ microsecond (HIRES80). The patterns are shifted out LSB first, so the LSB represents the leftmost dot position on the screen. This is the reverse of the way you visualize the numerical representation of the stored pattern (numbers have the LSB on the right).

Bit 7 of stored HIRES patterns is the delay bit. If this bit is set in HIRES40 processing, the entire 7-dot pattern is delayed by one 14M period (one fourth a COLOR REFERENCE period). This delay results in a very slight shift right, and consequent change in coloring, of the 7-dot pattern on the screen. Without the delay, HIRES40 patterns are black, white, green, and violet. With the delay, HIRES40 patterns are black, white, orange, and blue.

In HIRES80 processing there is no delay feature—it is not necessary because, with the higher resolution, the delay colors and other colors can be generated. Display memory bit 7 has no effect in HIRES80 mode.

VIDEO GENERATOR HARDWARE

The video generator consists of circuits that monitor the video scanner, the display mode soft switches, and the data driven from the display map in RAM to produce the VIDEO signal. This includes a considerable amount of logic circuitry in the IOU plus the video ROM, PICTURE signal load/shift register, and video summing amplifier. Figure 8.5 is a schematic diagram of these circuits.

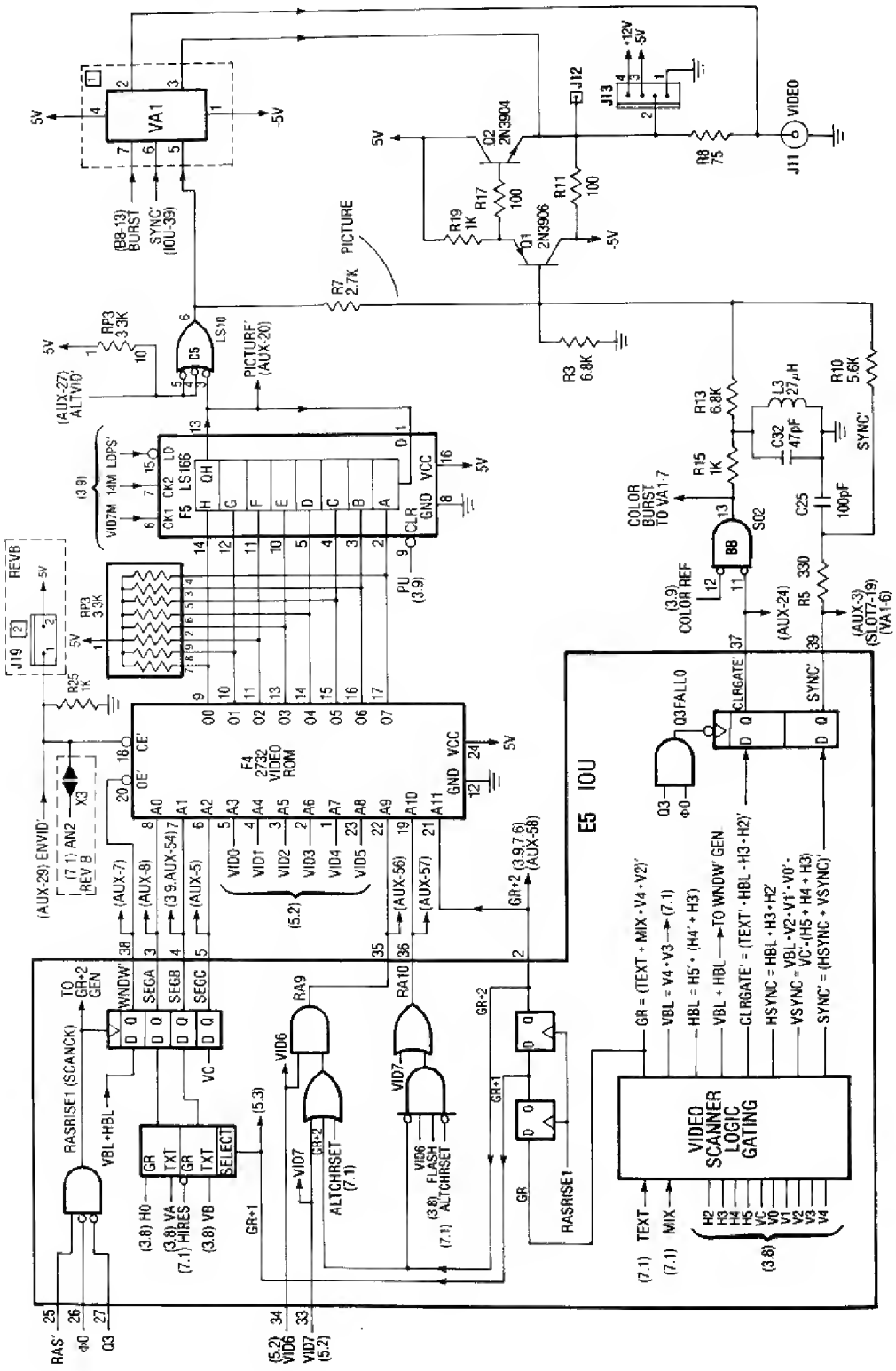
A key concept in Apple IIe video generation is that of a video counter scanning memory to drive out a display map in synchronization with the scanning of an electron beam across the face of a CRT. In Chapter 5, it was shown how the **video scanner** outputs are addressing inputs to motherboard and auxiliary card RAM during PHASE 1. In this chapter, it will be seen that all facets of video generation are performed in synchronization with the video scanner. This is accomplished by generating signals as a logical function of the video scanner inside the IOU. These scanner gated signals perform various functions in the video generation task such as blanking the picture, gating the COLOR BURST, and serving as the sync portion of the VIDEO signal.

A second key concept in Apple IIe video generation is that of the **video ROM**, translating video data from the display map in RAM to dot patterns which are shifted out to become the PICTURE signal. Every half cycle, when PHASE 0 rises or falls, new video data is present at the address inputs of this ROM. Depending on the display mode and the current state of the video scanner, the video ROM translates the data on the video data bus (VID0—7) to graphics or text patterns. Then the video patterns are loaded and shifted out by the **load/shift register**. In SINGLE-RES modes, the patterns resulting from auxiliary card RAM data are ignored and the patterns resulting from motherboard RAM data are loaded and shifted out: seven HIRES dot positions, one LORES block, or one TEXT character per microsecond. In DOUBLE-RES modes, patterns resulting from both motherboard and auxiliary card RAM data are loaded and shifted out: 14 HIRES dot positions, two LORES blocks, or two TEXT characters per microsecond.

VID0—VID5 are direct addressing inputs to A3—A8 of the video ROM. VID6 and VID7 are indirect addressing inputs, processed first in the IOU to implement FLASHING text and the functions of the ALTCHRSET soft switch. Processed VID6 and VID7 are labeled RA9 and RA10 (ROM Address 9 and 10) and connected to the A9 and A10 address inputs to the video ROM. The other address inputs to the video ROM are SEGA, SEGB, SEGC, and GR+2 from the IOU. Additionally, WNDW′ from the IOU and ENVID′ from the auxiliary slot are chip enable inputs to the video ROM.

The control inputs to the video ROM which come from the IOU are gated by video scanner states. These signals are delayed by one or two video scanner clocks (RAS′ rising during PHASE 1) for overall timing alignment. For example, SEGC is simply VC from the video scanner, delayed one scanner clock. The reason for the delay is that it takes time for data from video scanner access to RAM to become valid at VID0—VID7. The delay in the video scanner signals causes video data and control signals from the same video scanner state to be present at the video ROM at the same time. This is particularly important in the case of LORES graphics since H0 patterns are different than H0′ patterns.

The switching related to GRAPHICS/TEXT time switching in MIXED mode is delayed by two scanner clocks. This is so that the last GRAPHICS pattern at



NOTES:
[1] Connections provided for alternate video amplifier.

[2] Connections provided for J19 (remote ENVID' switch) in REV B.

Figure 8.5 Schematic: Apple IIe Video Generation.

the lower right of the display window will be entirely shifted out before the timing HAL is configured for TEXT shifting. MIXED mode switching is covered in a later section of this chapter.

It is normal to think of horizontal scan lines as beginning with horizontal sync, but, in the Apple, it is logical to think of horizontal scan lines as beginning with the first cycle of HBL. This is because HBL begins when the horizontal portion of the video scanner is preset and when the vertical portion increments. Signals gated by the video scanner tend to switch just after the display period ends. So as not to confuse it with the television horizontal scan, the 65-cycle period beginning with the horizontal preset will be referred to in the following discussion as the horizontal period.

Inputs to the Video ROM

ENVID' is a pulled down line connected from pin 29 of the auxiliary slot to an enable input of the video ROM. The data output lines of the video ROM are pulled up, so disabling the ROM yields high data which represents a black picture. An auxiliary card can, therefore, blank the motherboard PICTURE signal by bringing ENVID' high. Also, an auxiliary card can inject an alternate inverted picture signal to the motherboard on ALTVID'.

The fact that a high level out of the video ROM represents a black level on the screen is simply a matter of hardware convenience. "Black equals high" polarity here makes it easy to blank the display via pull-up resistors, and results in proper polarity of dot patterns at the video summing amplifier. A side effect is that dot patterns in the video ROM are inverted from the polarity in which GRAPHICS patterns are stored in the RAM display map. In HIRES or LORES patterns stored in RAM, "1" represents illumination and "0" represents blanking.

The ENVID' line is also connected through the X2 jumper to an addressing input of the keyboard ROM on Revision B motherboards, so pulling ENVID' high also selects the alternate keyboard layout (see Figure 7.4). Additionally, you can install a jack at the J19 location and plug an ENVID' switch into it, or you can solder jumper X3 and control ENVID' from a program via AN2.

WNDW' is the blanking gate of the Apple IIe, the signal which blanks the picture to cause the black margin around the display window. It goes high whenever HBL or VBL inside the IOU goes true (plus one scanner clock). WNDW' is connected from IOU-38 to the other enable input of the video ROM. Like the ENVID' line, when WNDW' is high, the video ROM is disabled and a black PICTURE signal results.

The HBL and VBL signals are not outputs of the IOU. I only deduce their existences because the logical way to develop WNDW' is as a combination of HBL and VBL, generated respectively from states of the horizontal and vertical portions of the video scanner. Also, WNDW' is true at exactly the same scanner states as the HBL + VBL BLANKING gate of the original Apple II (plus one scanner clock). On an oscilloscope, WNDW' is seen to consist of 192 alternating negative (40 scanner states) and positive (25 scanner states) levels followed by a very long positive level (4550 scanner states). The long positive level is the vertical blanking period, and the short duration alternations are the alternating horizontal display and horizontal blanking periods.

VID0—VID5 are address inputs to the video ROM as mentioned previously. The video ROM translates VID0—VID5 (and VID6—VID7, present via RA9 and RA10) to dot patterns for loading and shifting to the PICTURE signal. The general process is for the video to translate data from the display map on VID0—VID7 to dot patterns as dictated by other address inputs (GR+2, SEGA, SEGB, and SEGC).

The **GR** signal in Figure 8.5 is not the reset state of the TEXT/GRAPHICS soft switch. Rather it represents GRAPHICS time, which is all of the time in GRAPHICS NOMIX mode and all times except V4 • V2 of GRAPHICS MIXED mode. V4 • V2 identifies TEXT time in MIXED mode, and it is true during the last 32 horizontal periods of VBL' and during the last 38 undisplayed horizontal periods of VBL. This means that in MIXED mode the Apple switches to GRAPHICS then back to TEXT during VBL, but it is not significant because the screen is blank during VBL.

GR+1 and **GR+2** are the GR signal, delayed by one and two scanner clocks respectively. GR+1 is used in identifying HIRES time for scanner addressing (Figure 5.3) and in signal selection for SEGA and SEGB. GR+2 identifies GRAPHICS time in RA9 and RA10 generation and is the GRAPHICS time output of the IOU. As an addressing input, GR+2 divides the video ROM into GRAPHICS patterns and TEXT patterns.

GR+2 is also a direct input to the timing HAL on Revision A motherboards where it inhibits DOUBLE-RES timing during GRAPHICS time. In Revision B, GR+2 is inverted and gated by FRCTXT' before application to the timing HAL. If FRCTXT' is pulled low (normally by resetting AN3 with a 64K RAM card installed in the auxiliary slot), gated

GR+2′ is forced high. This enables DOUBLE-RES timing in the HAL if 80COL is set.

SEGA, **SEGB**, and **SEGC** are addressing inputs to the video ROM which subdivide the ROM differently depending on whether it is TEXT time or GRAPHICS time as identified by GR+1. The SEGA, SEGB, and SEGC signals are selected as listed in Table 8.1, then delayed one scanner clock before leaving the IOU.

The TEXT time assignments are such that SEGA, SEGB, and SEGC determine which of eight segments of a 7 x 8 text pattern is being drawn. Recall that VA, VB, and VC are not addressing inputs to RAM in TEXT/LORES scanning so the same 40 bytes of display memory are scanned eight times consecutively for each line of text or pair of LORES block rows. The assignment of VA, VB, and VC during TEXT time means that SEGA, SEGB, and SEGC determine which of eight segments of a 7 x 8 text pattern is to be driven out of the video ROM. Since these text segment address lines are connected to the three least significant address inputs of the video ROM, you will find the eight segments of each character pattern stored adjacently if you examine the text areas of the video ROM.

During GRAPHICS time, there is no need to identify each segment of the text lines, but there is need for other identification. VC is still needed to determine whether it is time to display the upper or lower row of LORES blocks. The LORES VC′ area of the video ROM contains a map of the patterns on VID0—VID3, and the LORES VC area contains a map of the patterns on VID4—VID7. As an example, if VID7—VID0 descending is 1011 1000, the LORES, VC′, H0′ output of the video ROM is 0111 0111 (the inversion of the pattern in VID3—VID0). The LORES, VC, H0′ pattern is 0100 0100. It will be seen later that the shift/load register rotates* these LORES patterns at a 14 MHz rate, effectively shifting the 4-bit pattern to the PICTURE signal 3.5 times in a LORES40 cycle.

The presence of H0 as an addressing input satisfies LORES requirements. Recall from Chapter 3 that the beginning phase of the 3.58 MHz COLOR REFERENCE alternates 180 degrees every MPU cycle, and that this phase can be defined in terms of H0. The color of a GRAPHICS pattern is different when driven from an even location than it is when driven from an odd location if no correction is made for the alternating phase of COLOR REFERENCE. There is no correction in HIRES, so even patterns are of different color than odd patterns. LORES GRAPHICS is corrected, and that is why H0 is an addressing input to the video ROM during GRAPHICS time.

*In this chapter, the term "rotate" is used in the same sense that "rotating" bits is used in 6502 assembly language.

Table 8.1
SEGA, SEGB, SEGC Signal Assignments*

	SEGA	SEGB	SEGC
GR+1′	VA	VB	VC
GR+1	H0	HIRES′	VC

*Selected signals are delayed one scanner clock before being output from IOU.

The correction consists of storing LORES patterns shifted two bits in the H0 areas of the video ROM. Continuing the above example with 1011 1000 on the video data bus, the LORES, VC′, H0 pattern is 1101 1101 (1000 inverted and rotated two bits). The LORES, VC, H0 pattern is 0001 0001 (1011 inverted and rotated two bits). The 2-bit bias is equivalent to ½ cycle of COLOR REFERENCE which is the required correction.

The third addressing input during GRAPHICS time is HIRES′, the inversion of the HIRES soft switch. This is needed at the video ROM because HIRES 7-dot patterns must be routed through the video ROM without regard to VC and H0. If you examine the HIRES GRAPHICS areas of the video ROM, you will find that there is no difference between VC′ and VC patterns or H0′ and H0 patterns.

The SEGB output of the IOU is also routed to the timing HAL where it is used to distinguish between HIRES and LORES when gated GR+2′ is low (GRAPHICS time). The timing HAL needs to identify HIRES40 GRAPHICS so it can delay HIRES timing by one 14M period if VID7 is high. This is the way the HIRES delayed feature is implemented in the Apple IIe.

The HIRES patterns stored in the video ROM are simple inversions of the patterns on VID6—VID0. The state of VID7 does not affect the HIRES pattern from the ROM because VID7 is the HIRES delay bit, not the on/off condition of a dot. As it is with H0 and VC, the VID7′ and VID7 portions of the HIRES area of the video ROM are identical. Similary, the MSB of HIRES patterns from the video ROM is not used. In the video ROM supplied with the Apple IIe, the MSB is set on all HIRES patterns.

The remaining address inputs to the video ROM are **RA9** and **RA10**. Like SEGA, SEGB, and SEGC, the functions of RA9 and RA10 depend on whether it is GRAPHICS or TEXT time. During GRAPHICS time as identified by GR+2, and during TEXT time

when ALTCHRSET is set, RA9 and RA10 are equivalent to VID6 and VID7, respectively (with IOU propagation delay). During GRAPHICS time, this allows the video ROM to translate HIRES and LORES VID7—VID0 GRAPHICS patterns into inverted patterns as described above, with direct map through of HIRES patterns and VC/H0 variations of LORES patterns.

Table 8.2 shows the TEXT and GRAPHICS time signal assignments to RA10 and RA9. Note that RA10 and RA9 are equivalent to VID7 and VID6 except during TEXT time when ALTCHRSET is reset. The special logic is necessary to implement FLASHING text, and because there are differences in the character code when ALTCHRSET is reset. Based on the Table 8.2 signal assignments, the TEXT time RA9 and RA10 states for various states of VID6, VID7, and ALTCHRSET are compiled in Table 8.3.

There is logic to the RA10 and RA9 states, but it is a little involved. For starters, each TEXT character is represented in the Apple display map by an 8-bit word. This means 256 possible character representations can be present on VID7—VID0 in TEXT processing. However, there are only 128 characters in a standard ASCII set, the control, special/numeric, upper case, and lower case groups. As a result, two complete alternate sets of ASCII can be represented in the TEXT display map. Additionally, the ALTCHRSET soft switch enables a programmer to select between two methods of interpreting the TEXT display code.

The original Apple II displayed only 64 upper case alphabetic, numeric, and special characters represented by the lower 6 bits of video data from the display map. The two most significant bits were interpreted by the video generator to display the characters as NORMAL, INVERSE, or FLASHING text. The Apple IIe TEXT display is compatible with the old Apple II format when ALTCHRSET is reset. The only difference is that the complete 128 NORMAL character set can be displayed, not just the upper case, special set.

When ALTCHRSET is set, the video data is interpreted as 128 NORMAL characters and 128 INVERSE characters, instead of 128 NORMAL, 64 INVERSE, and 64 FLASHING characters. Stored code for INVERSE characters is identical to that of NORMAL characters except VID7 is high for NORMAL characters and low for INVERSE characters.

Table 8.4 shows the characters of the video ROM that correspond to code stored in the display map. The ALTCHRSET layout seems far more logical since it is simply a complete INVERSE set and a complete NORMAL set. In fact, the TEXT area of the video ROM is exactly like the ALTCHRSET portion of Table 8.4. That is, RA10, RA9, and VID5 divide the TEXT area of the video ROM as follows:

RA10	RA09	VID5	CHARACTER SET
0	0	0	INVERSE control (upper)
0	0	1	INVERSE special
0	1	0	INVERSE upper
0	1	1	INVERSE lower
1	0	0	NORMAL control (upper)
1	0	1	NORMAL special
1	1	0	NORMAL upper
1	1	1	NORMAL lower

There are no FLASHING text patterns stored in the video ROM. Flashing text is mechanized by switching the RA10 and RA9 address between 00 and 10 for that video data with VID7 low and VID6 high when ALTCHRSET is reset. Switching between 10 and 00 alternately selects NORMAL control/special and INVERSE control/special characters. The control characters in the Apple IIe video ROM are identical to uppercase alphabetic characters, so the flashing text switches between INVERSE and NORMAL upper case, numeric, and special characters. The FLASH signal toggles once for every 16 video scanner overflows (Figure 3.8), so FLASHING text switches back and forth at a 1.87 Hz rate (3.74 alternations per second).

ALTCHRSET' INVERSE and FLASHING text is coded differently than NORMAL text and ALTCHRSET INVERSE text. VID5 is high for uppercase and low for special and numeric characters with ALTCHRSET' INVERSE and FLASHING text, but VID5 is low for uppercase and high for special and numeric characters with NORMAL and ALTCHRSET INVERSE text. As a result, video output routines should make appropriate

Table 8.2 RA9, RA10 Signal Assignments.

GR+2	ALTCHRSET	RA10	RA9
1	X	VID7	VID6
0	0	VID7 + VID6 • FLASH	VID6 • VID7
0	1	VID7	VID6

Table 8.3 RA9, RA10 TEXT Time States.

ALTCHRSET	VID7	VID6	RA10	RA9	CHARACTERS
0	0	0	0	0	INVERSE control/special
0	0	1	FLASH	0	Flash INVERSE/NORMAL*
0	1	0	1	0	NORMAL control/special
0	1	1	1	1	NORMAL upper/lower
1	0	0	0	0	INVERSE control/special
1	0	1	0	1	INVERSE upper/lower
1	1	0	1	0	NORMAL control/special
1	1	1	1	1	NORMAL upper/lower

*Flashes between NORMAL control/special and INVERSE control/special.

transformations to store correct code for desired INVERSE or FLASHING characters in the display area. The 40-column firmware of the Apple IIe does not correctly output lower case INVERSE video for display with ALTCHRSET set because it was written for the Apple II, not the Apple IIe. The 80-column firmware supports ALTCHRSET text, but it outputs INVERSE characters correctly only if ALCHRSET is set. In other words, use the 80-column firmware to output lower case INVERSE text from your Applesoft programs, but use the 40-column firmware if you wish to mix INVERSE and FLASHING text.

Loading and Shifting of Dot Patterns

For each 1-microsecond period between rising edges of PHASE 0, there are two 500-nanosecond accesses to the video ROM. Video data from the auxiliary card becomes valid on the video data bus shortly after PHASE 0 rises and remains there until shortly after PHASE 0 falls. Also, shortly after PHASE 0 falls, video data from the same motherboard RAM address becomes valid and remains valid until shortly after PHASE 0 rises. The pattern from the video ROM takes no more than 450 nanoseconds to become valid after its address is valid (assuming Apple uses a 450 nsec ROM), so the auxiliary card pattern is valid toward the end of PHASE 0, and the motherboard pattern is valid toward the end of PHASE 1.

The output of the video ROM is connected to a 74LS166 **load/shift** register that loads the dot patterns and shifts them out. The patterns are loaded when LDPS' drops low near the end of PHASE 0 (if DOUBLE-RES) and near the end of PHASE 1 (always). When the 8-bit load/shift register is not loading data, it is rotating the data in an end around shift. The rotation is only important in LORES40 mode, because new data is always loaded before the old data is completely rotated in the other modes.

The loading and shifting of dot patterns is controlled by three signals from the timing generator, LDPS', VID7M, and 14M. LDPS' is the load/shift control as noted before, 14M is the load/shift register clock, and VID7M serves as the clock enable (since it slightly lags 14M). The register loads or shifts data every time 14M rises when VID7M is low. The timing generator brings VID7M continuously low during DOUBLE-RES modes and LORES40 mode to enable 14 MHz processing. During TEXT40 and HIRES40 modes, VID7M is low every other 14M rising, so processing is at 7 MHz.

Note that auxiliary patterns appear at the input lines of the shift register every 500 nanoseconds, regardless of the Apple IIe display mode (assuming an auxiliary RAM card is installed). However, in SINGLE-RES mode processing, the timing generator does not bring LDPS' low at the end of PHASE 0, so the auxiliary RAM pattern is not processed. Display mode control is thus maintained via timing generator signals. More details of these timing signals will be discussed in following sections.

The QH output of the load/shift register is connected to the PICTURE' line, so the PICTURE' line rises and falls as a function of the rotating dot patterns. If QH is low or if the ALTVID' line from the auxiliary slot is low, the PICTURE signal goes high, a level that results in illumination on the television screen. The least significant bit from the video ROM is connected to the QH input of the load/shift register, so it is logical to think of QH as the least significant bit. It can then be clearly stated that **patterns are shifted with inversion to the PICTURE signal, LSB first.**

The PICTURE signal is applied to the video summing amplifier through a resistor that biases it

Table 8.4 Displayed Text Characters.

ALTCHRSET'

	INVERSE				FLASH				NORMAL							
	UPPER		SPECIAL		UPPER		SPECIAL		CONTROL		SPECIAL		UPPER		LOWER	
ASCII	000 $00	016 $10	032 $20	048 $30	064 $40	080 $50	096 $60	112 $70	128 $80	144 $90	160 $A0	176 $B0	192 $C0	208 $D0	224 $E0	240 $F0
0 $0	@	P		0	@	P		0	@	P		0	@	P	`	p
1 $1	A	Q	!	1	A	Q	!	1	A	Q	!	1	A	Q	a	q
2 $2	B	R	"	2	B	R	"	2	B	R	"	2	B	R	b	r
3 $3	C	S	#	3	C	S	#	3	C	S	#	3	C	S	c	s
4 $4	D	T	$	4	D	T	$	4	D	T	$	4	D	T	d	t
5 $5	E	U	%	5	E	U	%	5	E	U	%	5	E	U	e	u
6 $6	F	V	&	6	F	V	&	6	F	V	&	6	F	V	f	v
7 $7	G	W	'	7	G	W	'	7	G	W	'	7	G	W	g	w
8 $8	H	X	(	8	H	X	(	8	H	X	(	8	H	X	h	x
9 $9	I	Y	)	9	I	Y	)	9	I	Y	)	9	I	Y	i	y
10 $A	J	Z	*	:	J	Z	*	:	J	Z	*	:	J	Z	j	z
11 $B	K	[	+	;	K	[	+	;	K	[	+	;	K	[	k	{
12 $C	L	\	,	<	L	\	,	<	L	\	,	<	L	\	l	|
13 $D	M	]	-	=	M	]	-	=	M	]	-	=	M	]	m	}
14 $E	N	^	.	>	N	^	.	>	N	^	.	>	N	^	n	~
15 $F	O	_	/	?	O	_	/	?	O	_	/	?	O	_	o	※

ALTCHRSET

	INVERSE								NORMAL							
	UPPER		SPECIAL		UPPER		LOWER		CONTROL		SPECIAL		UPPER		LOWER	
ASCII	000 $00	016 $10	032 $20	048 $30	064 $40	080 $50	096 $60	112 $70	128 $80	144 $90	160 $A0	176 $B0	192 $C0	208 $D0	224 $E0	240 $F0
0 $0	@	P		0	@	P	`	p	@	P		0	@	P	`	p
1 $1	A	Q	!	1	A	Q	a	q	A	Q	!	1	A	Q	a	q
2 $2	B	R	"	2	B	R	b	r	B	R	"	2	B	R	b	r
3 $3	C	S	#	3	C	S	c	s	C	S	#	3	C	S	c	s
4 $4	D	T	$	4	D	T	d	t	D	T	$	4	D	T	d	t
5 $5	E	U	%	5	E	U	e	u	E	U	%	5	E	U	e	u
6 $6	F	V	&	6	F	V	f	v	F	V	&	6	F	V	f	v
7 $7	G	W	'	7	G	W	g	w	G	W	'	7	G	W	g	w
8 $8	H	X	(	8	H	X	h	x	H	X	(	8	H	X	h	x
9 $9	I	Y	)	9	I	Y	i	y	I	Y	)	9	I	Y	i	y
10 $A	J	Z	*	:	J	Z	j	z	J	Z	*	:	J	Z	j	z
11 $B	K	[	+	;	K	[	k	{	K	[	+	;	K	[	k	{
12 $C	L	\	,	<	L	\	l	|	L	\	,	<	L	\	l	|
13 $D	M	]	-	=	M	]	m	}	M	]	-	=	M	]	m	}
14 $E	N	^	.	>	N	^	n	~	N	^	.	>	N	^	n	~
15 $F	O	_	/	?	O	_	o	※	O	_	/	?	O	_	o	※

NOTES: ※ checkerboard

"Mouse text" replaces letters and characters at ASCII = $40—$5F in the enhanced firmware video ROM (see Chapter 6, **The Apple IIe Firmware Upgrade**).

to the correct voltage amplitude relative to SYNC' and COLOR BURST'. The three signals—PICTURE, SYNC', and COLOR BURST'—are summed together to make up the Apple IIe VIDEO signal of Figure 8.2. The VIDEO signal is routed from the summing amplifier to J12, J13, and J11, the video output jack at the back of the Apple.

SYNC' is a direct output of the IOU, generated from states of the video scanner. It is a combination of horizontal sync (a four scanner state pulse occurring in the middle of HBL) and vertical sync (four 56-state pulses occurring in the middle of VBL). For the American Apple, Figure 8.2 shows horizontal and vertical sync detail, Figure 8.5 shows the SYNC' logic equation, and the Figure 5.9 memory scanning map shows exactly when SYNC' occurs.

COLOR BURST' is a 14-cycle (four video scanner states) inverted sample of the COLOR REFERENCE signal. It is gated by **CLRGATE'** from the IOU which drops low immediately after horizontal sync if the TEXT soft switch is reset (see Figure 8.2). CLRGATE' never drops low if TEXT is set, so there is no COLOR BURST, and consequently no undesirable coloring of screen characters, in TEXT mode. Coloring of TEXT characters does occur in MIXED GRAPHICS mode because the COLOR BURST is not turned off, even during TEXT time.

SYNC' and CLRGATE' transitions do not occur quickly after the video scanner states, but appear to be clocked out by Q3 falling during PHASE 0. I speculate that this is done to remove gate width variations and switching spikes that can occur in logic gating. There is very good basis for this speculation because a switching spike on the vertical sync line causes a visible dark line in the left hand blanking margin of older Apple IIs. I'm sure that Apple is rightly wary of reintroducing a problem that they once solved.

Video Generation in Export Apples

The differences between foreign and American Apples are related primarily to video scanning and VIDEO signal generation. If not for television system incompatibility, the Apple IIe could be made to operate in any country by installing a power supply that would operate from the line voltage of that country. Supporting the special text requirements of the various languages is no problem because you can simply plug in a keyboard ROM and video ROM for any language.

The unfortunate reality is that there are several television systems used throughout the world, incompatible with each other to one degree or another. One area of incompatibility is the scanning rate. The NTSC standard scanning pattern is made up of 525 horizontal scans in two interlaced fields scanned at 60 fields per second. Other television systems have pictures made up of 625 horizontal scans in two interlaced fields scanned at 50 fields per second. Apple IIe compatibility with televisions and monitors designed to scan at the slower rate is achieved through installation of a 50 Hz IOU, as opposed to the 60 Hz IOU of American Apples.

In the 50 Hz IOU, the vertical portion of the video scanner presets to 011001000, fifty less than the 60 Hz 011111010. This adds 50 scans to the normal 262 for 312 horizontal scans in a 50 Hz Apple. All 50 of these scans are added to VBL so that VBL is 120 scans long instead of 70. Additionally, the vertical sync is shifted from that of the 60 Hz IOU so equal black margins are maintained at the top and bottom of the screen. Horizontal scanning and sync are the same in both IOUs.

The vertical sync equation in 50 Hz IOUs is VBL• V5' • V2' • V0' • VC' • (H5 + H4 + H3) as opposed to VBL • V2 • V1' • V0' • VC' • (H5 + H4 + H3) in 60 Hz IOUs. Vertical sync consists of four 56-state pulses, just as in American Apple scanning. These occur during horizontal periods 73, 74, 75, and 76 of VBL in 60 Hz Apple scanning. By way of comparison, 60 Hz IOUs have 36 horizontal periods in VBL up through vertical sync and 34 horizontal periods in VBL afterwards. 50 Hz IOUs have 76 horizontal periods in VBL up through vertical sync and 44 horizontal periods in VBL afterwards.

The critical states of VA—V5 in both 60 Hz and 50 Hz IOUs are summarized in Table 8.5. Old Apple hands may recognize that Apple IIe 60 Hz scanning

Table 8.5 American (60 Hz) and European (50 Hz) Scanning Differences.

	START VBL V543210CBA	VERTICAL SYNC V543210CBA	PRESET ON OVERFLOW V543210CBA	VERTICAL SYNC V543210CBA	END VBL V543210CBA
American	111000000	1111000XX	011111010	---	100000000
European	111000000	---	011001000	0110100XX	100000000

is identical to that of the RFI Revision Apple II, and that 50 Hz scanning is identical to that of the RFI Revision Apple II with 50 Hz jumpers made.

The 50 Hz scanning of foreign Apples is fine tuned by changing the 14 MHz crystal on motherboards in which 50 Hz IOUs are installed. The frequency is changed from 14.31818 to 14.25045 in motherboards with discrete circuit oscillators, and to 14.25 in motherboards with hybrid circuit oscillators (and in 50 Hz Apple IIc's). This is done so that the Apple IIe horizontal scan period is approximately equal to the 64-microsecond horizontal period of 50 Hz television systems.

Changing 14M slightly alters the 6502 execution speed (from 1.0205 MHz to 1.016 MHz) and reduces COLOR REFERENCE to 3.56 MHz. The 6502 speed reduction is apparently not enough to make disk I/O unreliable when reading American disks on 50 Hz Apples or vice versa. The reduction in COLOR REFERENCE frequency is not important because, while COLOR REFERENCE is the reference for color on the PICTURE' line in 50 Hz Apples, it is not of the frequency at which color is passed to a television or monitor.

The second important area of incompatibility in television systems is the different methods of representing color information. The three principle systems are NTSC, PAL (Phase Alternating Lines), and SECAM (SEquential Color And Memory). There are no Apple IIe SECAM adaptors, but the Apple IIe is used in some SECAM countries using RGB adaptors and monitors. This is especially true of France, whose television sets have an RGB input jack (the Peritel jack) in the back. PAL is the system used in European Economic Community countries (except France), and Apple competes in this important computer marketplace by manufacturing an Apple IIe PAL motherboard.

Figure 8.6 is a schematic of the circuitry that is added to an NTSC motherboard to make it a PAL motherboard. This figure is an Apple drawing, reproduced here with permission of Apple Computer, Inc. The notes in italics and the video ROM and keyboard ROM depictions were inserted by this author. The PAL circuitry is nearly identical to the circuitry of the Apple II "Eurocolor" card, so basically an Apple IIe PAL motherboard is an American motherboard with a 14.25 MHz oscillator, a 50 Hz IOU, foreign language video and keyboard ROMs, and a built-in Eurocolor card.

PAL video uses a 4.43619 MHz color subcarrier as opposed to the 3.579545 of NTSC, and the PAL COLOR BURST alternates 90 degrees in PHASE every horizontal scan. Televisions and monitors designed to display PAL video will not, therefore, automatically produce colored displays from 3.56 MHz Apple IIe dot patterns interspersed with 3.56 MHz REFERENCE BURSTS. To generate a signal that will produce a color display on a PAL monitor, the PAL circuitry extracts color information from the serial dot pattern and 3.56 MHz COLOR REFERENCE. Then a PAL VIDEO output signal is constructed with luminance signal, modulated 4.44 MHz color subcarrier, phase alternating 4.44 MHz COLOR BURST, and scanning sync.*

The dot patterns on the PICTURE' (SEROUT') line are shifted at 14 MHz to an LS164 serial in, parallel out shift register and sampled as 4-dot patterns, four dots being the number of HIRES80 dots in a COLOR REFERENCE cycle. Every time COLOR REFERENCE rises, a new 4-dot pattern is latched in an LS175 quad D flip-flop. The output of the four flip-flops is a 4-bit number which represents the phase relationship between the current 4-dot pattern and COLOR REFERENCE, in other words, the color of the Apple IIe dot pattern.

The 4-bit color word is converted through summing resistors to DC voltage levels representing weighted R−Y (red minus luminance) and B−Y (blue minus luminance) color signals. These are the color signals which are used to modulate a subcarrier in conventional PAL broadcast television. The color signals are summed with modulator balance voltages and burst gate voltages, and applied to the R−Y and B−Y reference inputs of a TCA650 chrominance demodulator.

The TCA650 is an IC that is intended for use as a synchronous demodulator in PAL or SECAM television sets. In the Apple IIe PAL circuitry, the TCA650 is used for exactly the opposite function, modulating a constant phase subcarrier with B−Y, modulating a phase alternated subcarrier with R−Y, and combining the two modulated subcarriers to form the chrominance signal. This is not a normal TCA650 application but an impressively innovative design effort by Gary Baker, the Apple engineer who designed the Eurocolor card. Rather than cause confusion by insisting that the TCA650 is a demodulator, I will refer to it as a modulator with modulating inputs at pins 6 and 7 and subcarrier inputs at pins 11 and 9.

The color subcarrier comes from a 4.43619 MHz oscillator and is applied to pin 1 of the TCA650. It is

*The 4.44 MHz signal should be referred to as a carrier to be technically correct. It becomes a subcarrier if the VIDEO signal modulates a television RF carrier. However, referring to this signal as the subcarrier helps to distinguish it from the RF carrier, and so shall it be here.

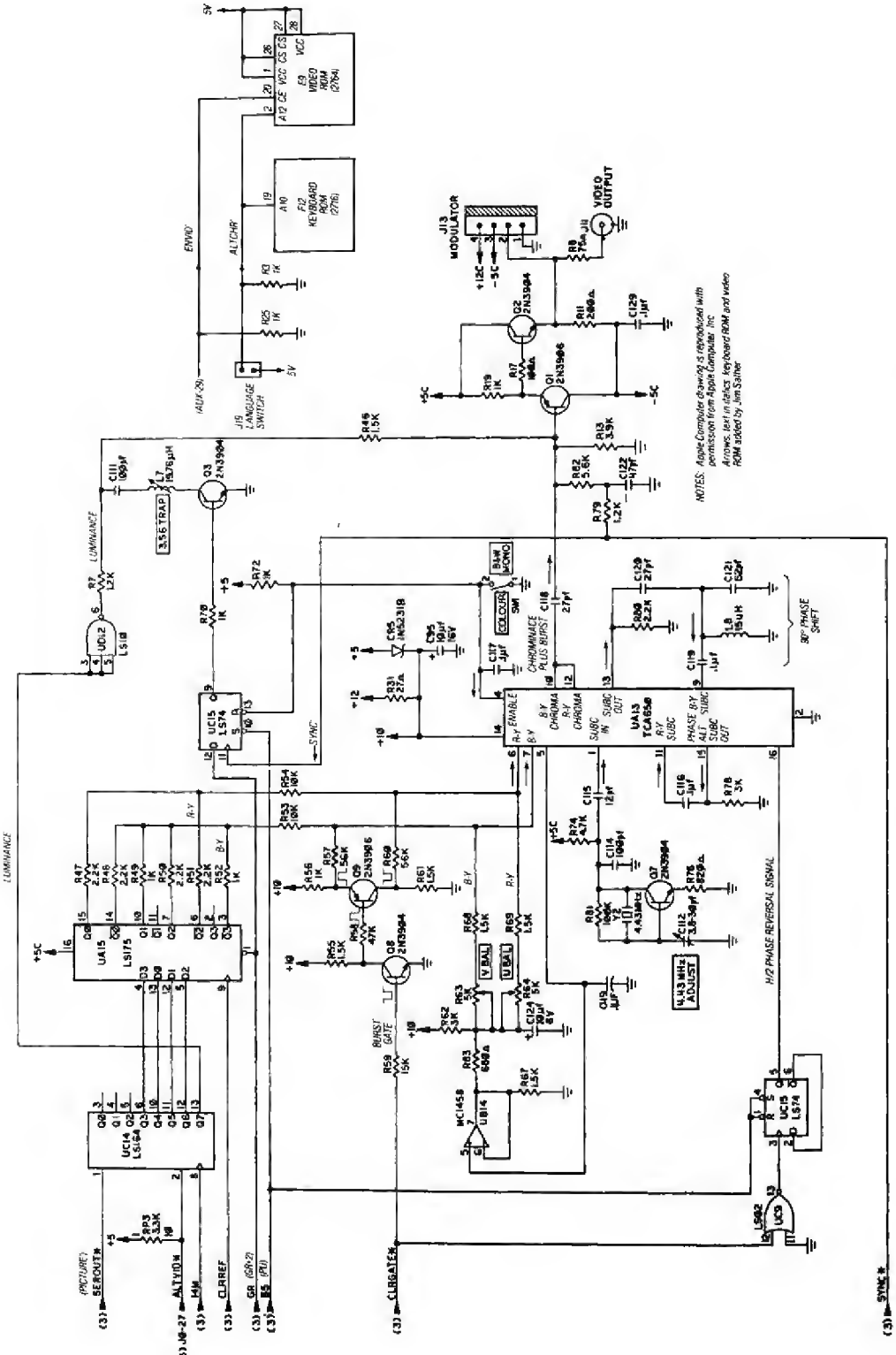


Figure 8.6 Schematic: PAL Motherboard Video Circuits.

output at pins 13 and 15 at equal amplitude. The constant phase pin 13 output is routed through 90 degree phase shifting circuitry to pin 9, the subcarrier input to the B−Y modulator. The pin 15 output alternates its phase as controlled by the pin 16 switching input. This input toggles when CLRGATE' falls, so the pin 15 signal alternates in phase every horizontal scan. The phase alternating output from pin 15 is coupled to pin 11, the subcarrier input to the R−Y modulator.

The 90 degree shifted, B−Y modulated subcarrier and the phase alternated, R−Y modulated subcarrier are combined together at pins 10 and 12 of the TCA650. This signal is a combined CHROMINANCE/BURST signal. The BURST is generated by level shifting the B−Y and R−Y signals in opposite polarity during CLRGATE' low. The modulation balance controls are adjusted so there is no chrominance signal out of the TCA650 when the 4-bit color word is 1111 and CLRGATE' is high. When CLRGATE' drops low, B−Y raises and R−Y lowers to cause a 4-microsecond sample of the 4.4 MHz subcarrier to appear at pins 10 and 12 of the TCA650. Combining the 180 degree phase alternated subcarrier and the 90 degree shifted subcarrier together in equal amplitude results in a COLOR BURST that alternates 90 degrees in phase every horizontal scan.

The serial dot pattern on PICTURE', delayed by the LS164 shift register, is inverted to form the LUMINANCE signal which is summed with the SYNC' and CHROMINANCE/BURST signals to produce the PAL VIDEO output signal. There is a gated trap which removes 3.56 MHz variations from the LUMINANCE signal during GRAPHICS time when the COLOUR/MONO switch is set to COLOUR. This is necessary to prevent 3.56 MHz LUMINANCE patterns from causing color interference since 3.56 MHz is within the bandwidth of the chrominance amplifier of a PAL monitor or television.

Trapping 3.56 MHz from the LUMINANCE signal causes 3.56 MHz dot patterns to be blurred. This enhances colored solids on the screen but prevents clear display of high resolution monochrome graphics. The COLOUR/MONO switch should, therefore, be set to MONO when the Apple IIe is used with a monochrome monitor. In addition to disabling the 3.56 MHz trap, this disables the CHROMINANCE/BURST signal from the TCA650.

Besides the PAL circuitry, there are differences in the keyboard and video ROMs of the PAL motherboard of which Americans can be justifiably envious. The video ROM is an 8 kilobyte, 28-pin 2764 compatible ROM instead of the 4 kilobyte, 24-pin 2732 compatible ROM of American Apples. A simple switch line (ALTCHR) connected to address inputs of the keyboard and video ROMs switches between American and foreign language text characters. With the big video ROM, there truly are complete alternate text character sets to choose from in the PAL Apple IIe.

There are some countries (Canada for example) which use the NTSC standard television system but still have foreign language requirements. To support the bilingual needs of these countries, Apple developed a 24- to 28-pin socket adaptor for the video ROM socket of the NTSC motherboard. When Apple begins to support a bilingual NTSC market like this, they use the adaptor to enable installation of 28-pin 2764 biligual video EPROMs. However, if the market gets big enough to justify it, they mask the bilingual patterns into a 24-pin, 8 kilobyte ROM. I am not certain how to obtain one of the adaptors in the U.S., but it would not be particularly difficult to build an adaptor like this if you desire to switch between displayed text character sets in an NTSC Apple IIe (see Figure 8.18).

DISPLAY MODE SOFT SWITCHES

The display mode of the Apple is set up by programmable soft switches in the IOU. The IOU soft switches are illustrated in Figure 7.1, and the functions of those soft switches related to the video display are summarized in Table 8.6.

There are three basic display modes, TEXT, LORES graphics, and HIRES graphics selected via the **TEXT** (GRAPHICS') and **HIRES** (LORES') soft switches. Additionally, MIXED displays of HIRES or LORES with four lines of text at the bottom of the screen may be selected by resetting TEXT and setting MIXED.

PAGE2 selects between the PAGE1 ($400—$7FF and $2000—$3FFF) and PAGE2 ($800—$BFF and $4000—$5FFF) scanned display areas if **80STORE** is reset. If 80STORE is set, PAGE2 performs the motherboard/auxiliary RAM access management functions described in Chapter 5.

The **80COL** soft switch performs only one function. It is inverted to 80COL' and output from the IOU to the timing HAL where it selects DOUBLE-RES timing if GR+2 is low in Revision A or gated GR+2' is high in Revision B. In other words, DOUBLE-RES timing is enabled when 80COL' is low during TEXT time or forced (by FRCTXT') TEXT time. On Revision B motherboards with a 64K RAM card with DOUBLE-RES GRAPHICS jumper installed, **FRCTXT'** is brought low to force gated

Table 8.6 Functions of the Display Mode Soft Switches.

SOFT SWITCH	OFF ADDRESS	ON ADDRESS	READ ADDRESS	FUNCTION
80STORE*	W$C000	W$C001	R$C018	Disable PAGE2 display area switching
80COL	W$C00C	W$C00D	R$C01F	DOUBLE-RES timing if TEXT time or AN3'
ALTCHRSET	W$C00E	W$C00F	R$C01E	Display full INVERSE ASCII text set
TEXT***	$C050	$C051	R$C01A	TEXT display
MIXED***	$C052	$C053	R$C01B	MIXED GRAPHICS/TEXT display if TEXT'
PAGE2*	$C054	$C055	R$C01C	Scan secondary RAM display area
HIRES*	$C056	$C057	R$C01D	HIRES display if GRAPHICS time
AN3**	$C05E	$C05F	NA	DOUBLE-RES GRAPHICS timing when AN3'

NOTES:
* PAGE2, HIRES, and 80STORE are mechanized identically in the MMU and IOU. The MMU passes the state of 80STORE to MD7 when $C018 is read, and the IOU passes the state of PAGE2 or HIRES to MD7 when $C01C or $C01D is read.
** AN3 is jumpered to the FRCTXT' (ENFIRM in Rev A) line on 64K auxiliary RAM cards. When AN3 is low, this forces gated GR+2' high at the timing HAL, enabling DOUBLE-RES timing if 80COL' is low.
*** All Table 8.6 soft switches except TEXT and MIXED are reset when RESET' falls.

GR+2' high by resetting AN3. On Revision A motherboards, there is no FRCTXT' line and no way to display DOUBLE-RES GRAPHICS.

Now that the video generator hardware has been introduced, there is a basis for understanding how soft switches control the display modes of the Apple IIe. First, they affect RAM addressing so that they control the area of RAM which is being scanned during PHASE 1. Second, they affect video ROM addressing so that they control the translation of video data to dot patterns. Third, they affect timing generation so that the PICTURE signal is loaded and shifted at SINGLE-RES or DOUBLE-RES rates. Table 8.7 shows the various display mode control soft switches and how they accomplish their functions.

The following programming examples should serve to illustrate the principles of display mode selection in the Apple IIe.

```
TEXT40, PAGE1
 LDA $C051    TEXT
 LDA $C054    PAGE2 OFF
 STA $C00C    80COL OFF (NO DOUBLE-RES)

TEXT80, PAGE2
 LDA $C051    TEXT
 LDA $C055    PAGE2
 STA $C00D    80COL (DOUBLE-RES)
 STA $C000    80STORE OFF (ENABLE
              PAGE2 SCAN)

HIRES40 MIXED WITH TEXT40, PAGE1
 LDA $C050    TEXT OFF (GRAPHICS)
 LDA $C053    MIXED
 LDA $C054    PAGE2 OFF
 LDA $C057    HIRES
 LDA $C05F    AN3 ON (ENABLE HIRES
              DELAYED)
 STA $C00C    80COL OFF (NO DOUBLE-RES)

HIRES80 GRAPHICS, NOMIX, PAGE1
 LDA $C050    TEXT OFF (GRAPHICS)
 LDA $C052    MIXED OFF
 LDA $C057    HIRES
 LDA $C05E    AN3 OFF (DOUBLE-RES
              GRAPHICS TIMING)
 STA $C001    80STORE (DISABLE PAGE2
              SCAN)
 STA $C00D    80COL (DOUBLE-RES)

LORES40 MIXED WITH TEXT80, PAGE2
 LDA $C050    TEXT OFF (GRAPHICS)
 LDA $C053    MIXED
 LDA $C055    PAGE2
 LDA $C056    HIRES OFF (LORES)
 LDA $C05F    AN3 ON (SINGLE-RES
              GRAPHICS TIMING)
 STA $C000    80STORE OFF (ENABLE PAGE2
              SCAN)
 STA $C00D    80COL (DOUBLE-RES)
```

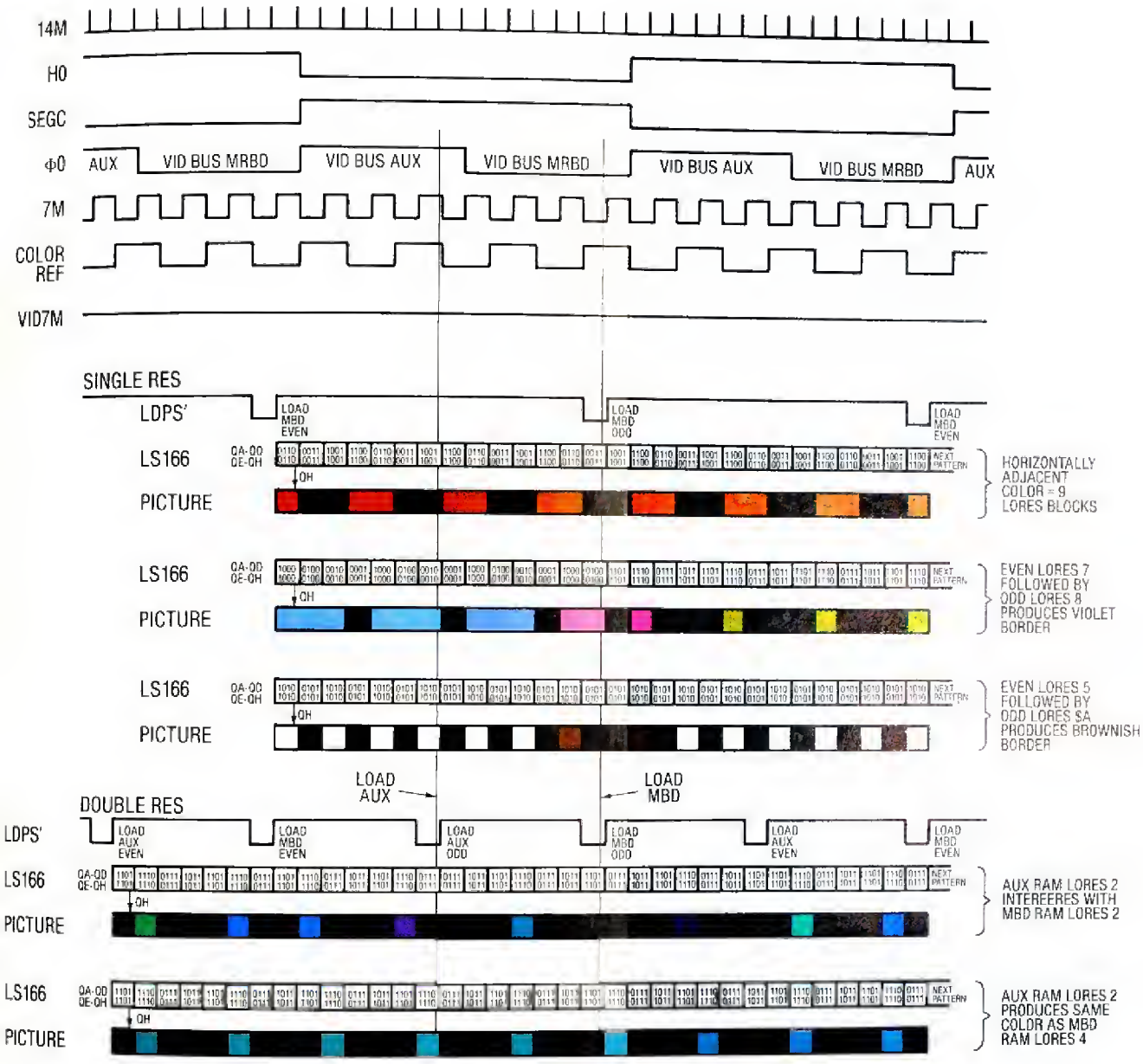


Figure 8.9 LORES Video Output.

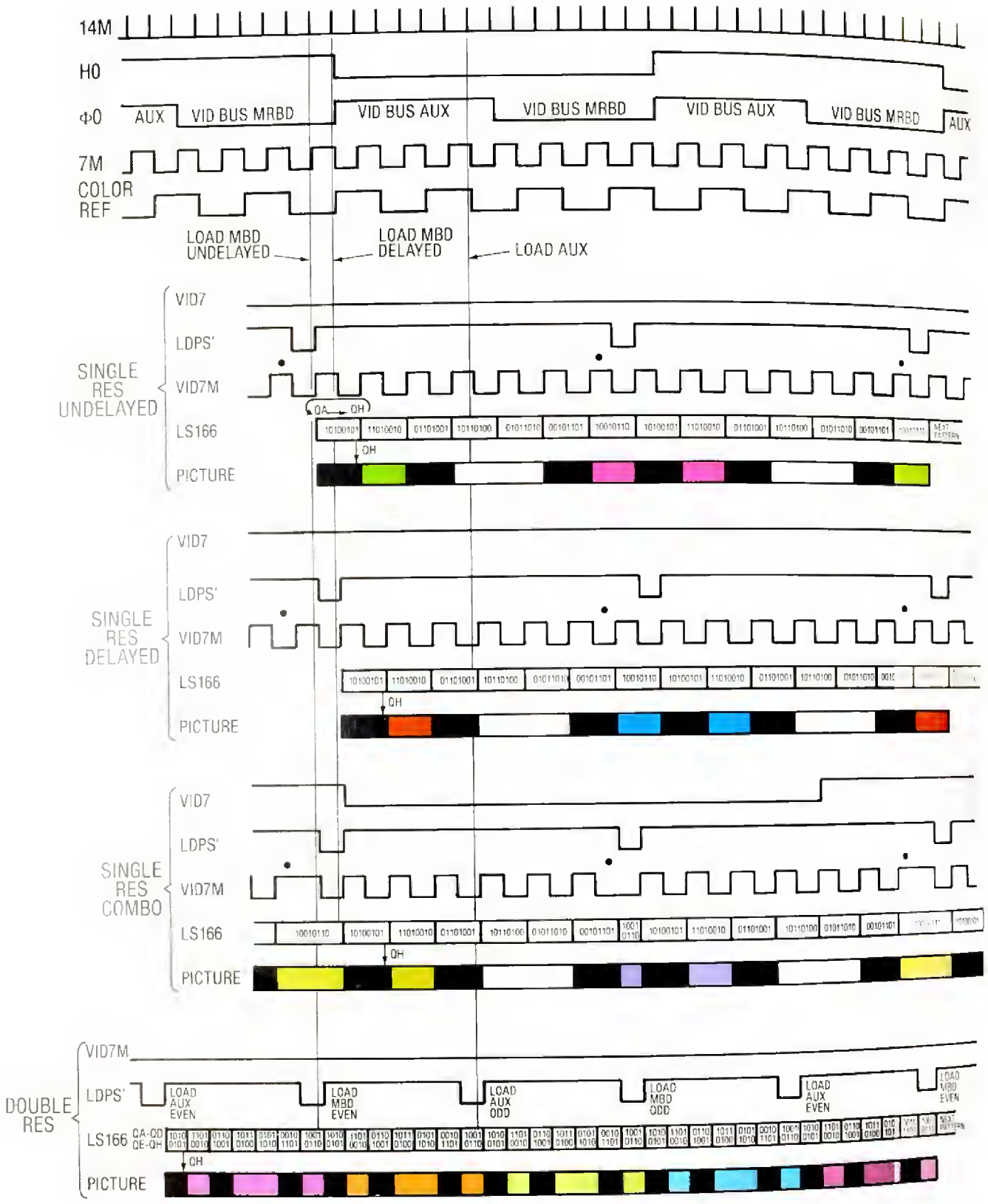


Figure 8.13 HIRES Video Output.

Figure 8.11 The LORES/HIRES80 Colors have a Cyclic Nature.

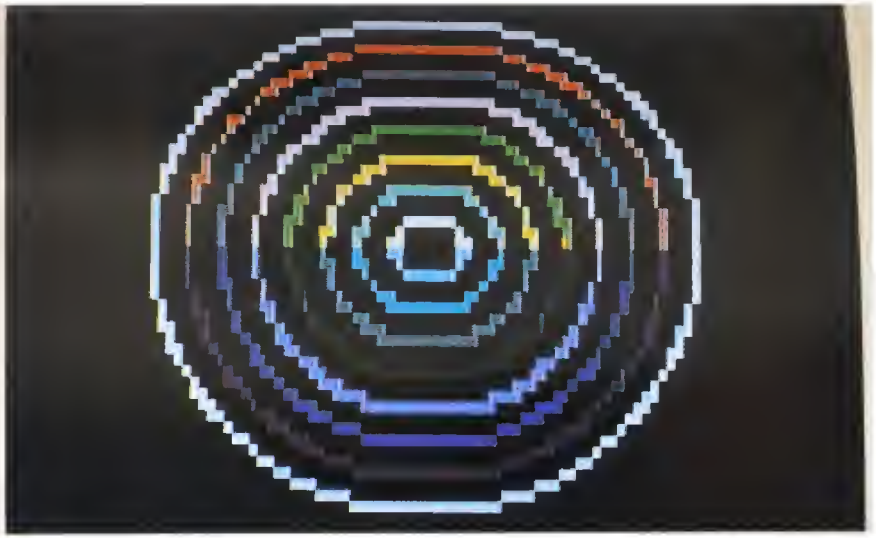


Figure 8.12 A LORES80 Display (from Figure 8.20 Program).

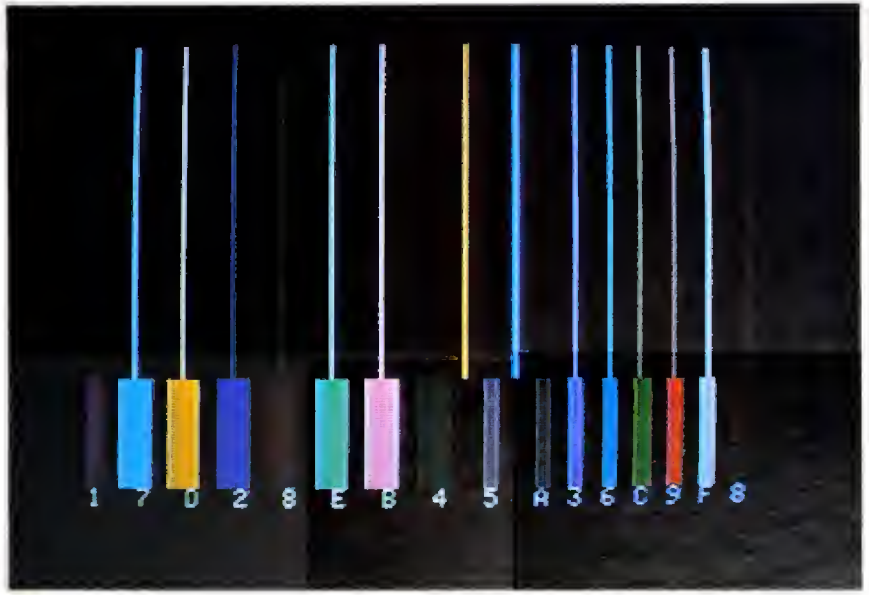


Figure 8.14 HIRES40 Pattern Interference Display (from Figure 3.11 Program).

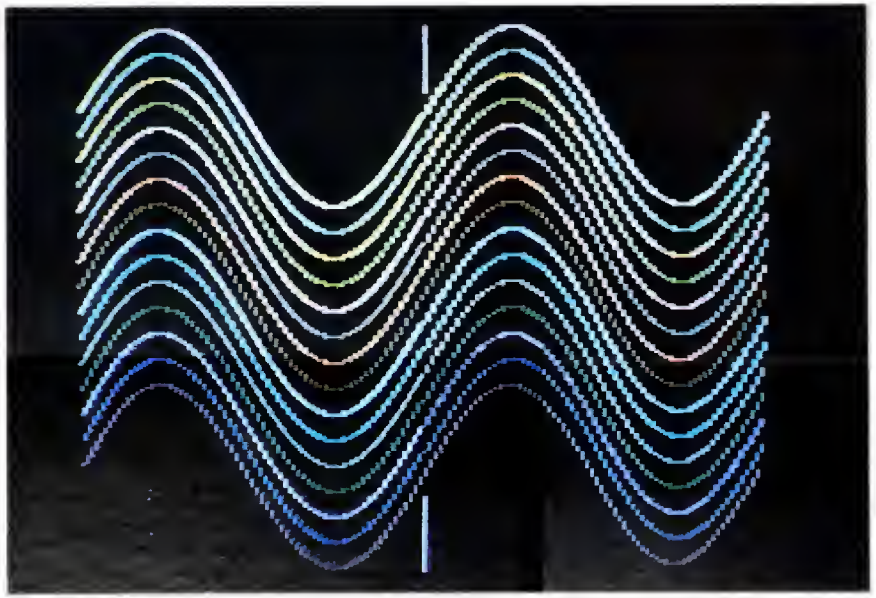


Figure 8.16 A HIRES80 Display (from Figure 8.21 Program).

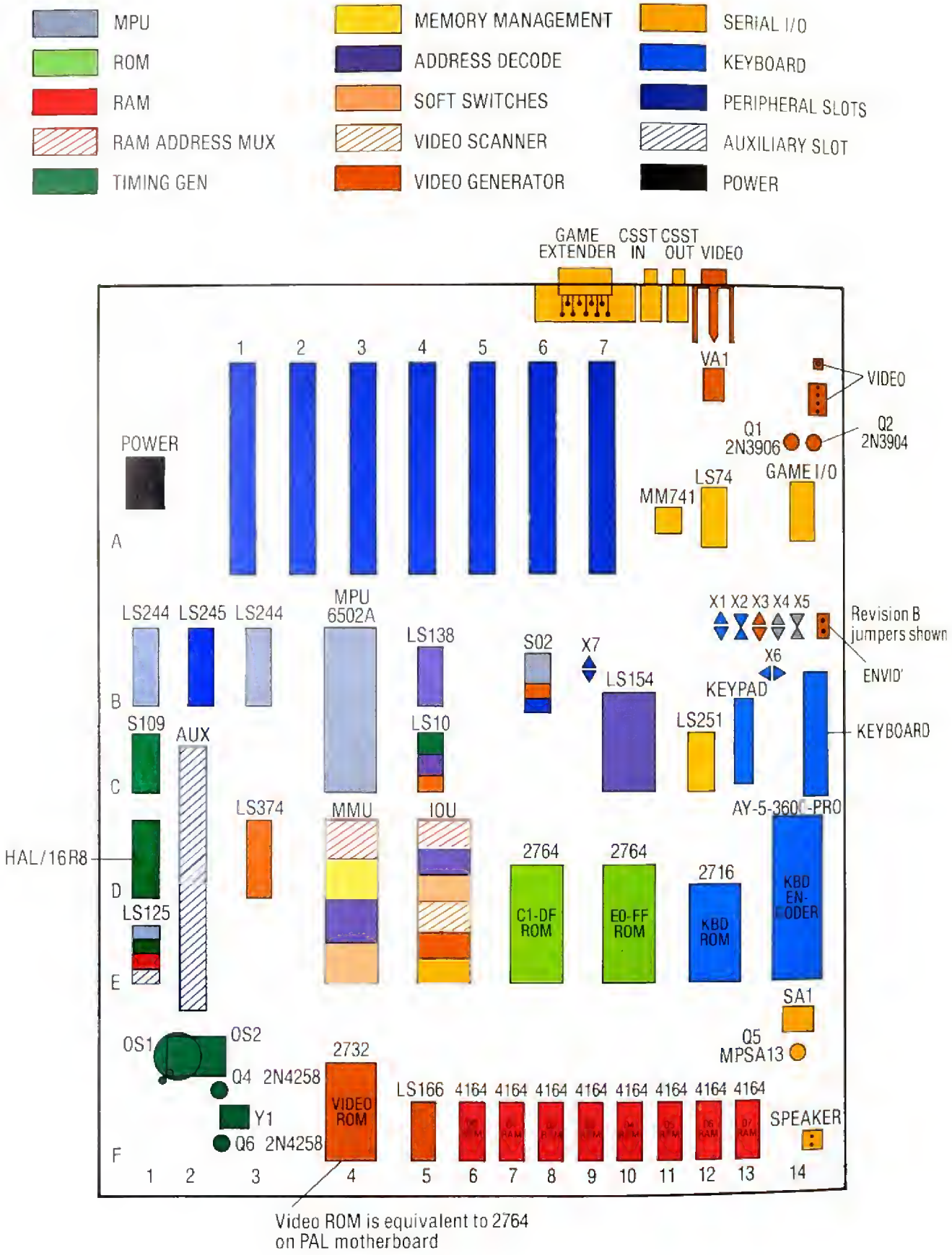


Figure 10.3 Motherboard Component Locations.

Table 8.7 Mechanization of Display Mode Soft Switches.

SOFT SWITCH	RESET FUNCTION	SET MECHANIZATION
TEXT	GRAPHICS	Scanning of $400—$BFF (Fig 5.3) Segmented addressing of text patterns at video ROM (Fig 8.5) INVERSE, NORMAL, FLASH interpretation of VID6, VID7 (Fig 8.5) Disable COLOR BURST via CLRGATE' (Fig 8.5) Enable DOUBLE-RES timing (Fig 3.9)
MIXED	NOMIX	Scanned address switched to TEXT at screen bottom (Fig 5.3) Video ROM address switched to TEXT at screen bottom (Fig 8.5) Enable DOUBLE-RES timing at screen bottom (Fig 3.9)
PAGE2	PAGE1	Scanning of $800—$BFF if TEXT/LORES, 80STORE' (Fig 5.3) Scanning of $4000—$5FFF if HIRES, 80STORE' (Fig 5.3) Aux RAM access at $400—$7FF if 80STORE (Fig 5.13b) Aux RAM access at $2000—$3FFF if 80STORE, HIRES (Fig 5.13b)
HIRES	LORES	Scanning of $2000—$5FFF if GR+1 time (Fig 5.3) HIRES video ROM address if GR+1 time (Fig 8.5) Monitor VID7 for delayed timing if SINGLE-RES, GR+2 (Fig 3.9) PAGE2 selects $2000—$3FFF main/aux RAM if 80STORE (Fig 5.13b)
FRCTXT'	FRCTXT	DOUBLE-RES GRAPHICS timing if 80COL (Fig 3.9) SINGLE-RES, 7MHz, undelayed timing if 80COL' (Figure 3.9)
80STORE	80STORE'	Disable PAGE2 display management functions (Fig 5.3) Enable PAGE2/HIRES memory management functions (Fig 5.13b)
80COL	40COL	Enable DOUBLE-RES TEXT or forced TEXT timing (Fig 3.9)
ALTCHR	NRMCHR	Enable lower case INVERSE video ROM addressing (Fig 8.5)

VIDEO GENERATION TIMING SIGNALS

The video generation timing signals are LDPS' (LoaD Parallel in, Serial out register) and VID7M (VIDeo 7 MHz). They are outputs of the timing HAL and were covered superficially in Chapter 3, but they are covered in detail here because they are related solely to the loading and shifting of video patterns.

As mentioned previously, LDPS' is the load/shift control signal for video output and VID7M is the load/shift clock ENABLE' signal. The video load/shift register loads or shifts when 14M rises while VID7M is low. If 14M rises while VID7M and LDPS' are low, the video pattern at the output of the video ROM is loaded into the load/shift register. If 14M rises while VID7M is low and LDPS' is high, the pattern currently residing in the load/shift register is rotated QH > QA, QA > QB, QB > QC, etc.

A number of the features of Apple IIe display modes are realized through variations of LDPS' and VID7M as shown in Table 8.8. These variations will be made more clear as the specific examples of Figures 8.7, 8.9, and 8.13 are discussed.

The logic equations of the timing HAL outputs are given in Table 3.4. The LDPS' and VID7M entries from Table 3.4 are repeated here in Table 8.9 for ease of reference. These equations are valid for HALs meant for use in Revision B motherboards. The Revision A HAL supports GR+2 input instead of GR+2', and there some differences in timing signal phase. Some important aspects of the Revision B VID7M and LDPS' equations follow here. References in parentheses such as (S1) refer to logic equations in Table 8.9.

1. Gated GR+2' is used to differentiate between TEXT time and GRAPHICS time. This signal is represented by GR' in the logic equations, and it is low during GRAPHICS time if FRCTXT' is high. GR' in the equations is true during TEXT or forced TEXT time. GR'' is true during GRAPHICS time when FRCTXT' is high.
2. SEGB is equivalent to LORES (HIRES soft switch reset) during GRAPHICS time, and it is used in the timing HAL to distinguish between LORES and HIRES GRAPHICS.

Table 8.8 Display Mode Variations of VID7M and LDPS'.

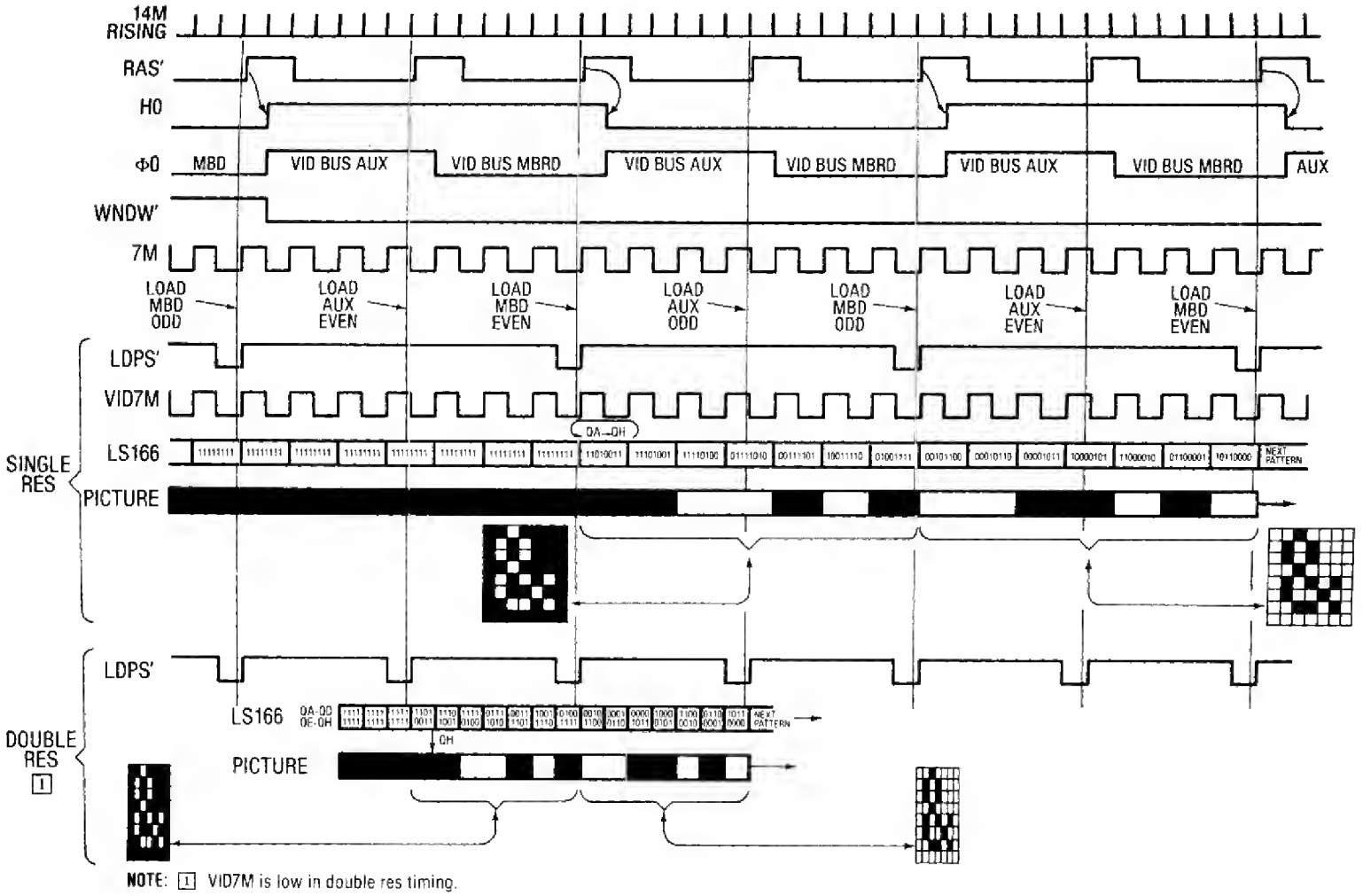
MODE	VARIATION	RESULT
TEXT40	VID7M = 7M LDPS' during PHASE 1	7 MHz load/shift One video cycle per MPU cycle
HIRES40*	VID7M = 7M if VID7' VID7M = 7M' if VID7 LDPS' during PHASE 1 LDPS' delayed if VID7	7 MHz load/shift Delayed 7 MHz load/shift One video cycle per MPU cycle Video cycle delayed if VID7
LORES40*	VID7M constant low LDPS' during PHASE 1	14 MHz load/shift One video cycle per MPU cycle
DOUBLE-RES MODES	VID7M constant low PHASE 1, PHASE 0 LDPS'	14 MHz load/shift Two video cycles per MPU cycle
*Forced undelayed 7 MHz processing if FRCTXT' is low.		

3. VID7M is constantly low during LORES GRAPHICS (S1). This causes 14M LORES processing at the load/shift register. An abnormal undelayed 7M LORES mode can be selected by resetting 80COL and bringing FRCTXT' low. VID7M is identical to 7M in this mode.
4. VID7M is constantly low during DOUBLE-RES (80COL, TEXT or forced TEXT) time (S2).
5. If 80COL is reset, VID7M is identical to 7M during TEXT or forced TEXT time (S3). This forces VID7M to be identical to 7M in TEXT40 mode. Incidental side effects are that FRCTXT' low forces undelayed processing of HIRES40, and the abnormal 7M LORES processing described in item 3.
6. Equations S1—S3 cover all VID7M processing except HIRES40, and equations S4—S5 and T1—T3 cover only HIRES40. At the end of PHASE 1 • Q3' • AX' (marked by dots in Figures 3.2 and 8.13) in HIRES40 processing, VID7M falls if VID7 is low and does not fall if VID7 is high (S4). VID7M toggles at all other points in HIRES40 processing (T1, T2, and T3). The fact that VID7M does not fall after PHASE 1 • Q3' • AX' when VID7 is high creates a 14M period delay in processing for the 7-dot HIRES pattern present on VID6—VID0. This is why setting bit 7 of a stored HIRES pattern results in blue or orange coloring instead of violet or green in SINGLE-RES processing.
7. VID7M is forced to fall after PHASE 1 • AX' • Q3' when processing the first blanked video cycle at the right side of the screen (S5). This, along with undelayed LDPS', cuts off the far right delayed dot on the HIRES40 screen.
8. LDPS' falls only during PHASE 1 of SINGLE-RES processing. In DOUBLE-RES processing, LDPS' falls after AX' • Q3' of both PHASE 0 and PHASE 1 (S1). In other words, in DOUBLE-RES processing, the auxiliary card RAM pattern is loaded, in addition to the motherboard RAM pattern.
9. During TEXT time (S2), forced TEXT time (S2), LORES time (S3), undelayed HIRES (S4), and at the right edge of the display screen (S5), LDPS' falls after PHASE 1 • AX' • Q3'. During a HIRES delayed cycle (S6), LDPS' falls one 14M period later causing a delayed load of the HIRES pattern.

The abnormal 7M LORES mode mentioned in items 3 and 5 is largely useless, and knowledge of its existence is not widespread. 7M LORES processing is basically the processing of LORES patterns at HIRES40 speed. It results in fragmented blocks for all patterns except 0000, 0101, 1010, and 1111 which result in blocks the same colors as the undelayed HIRES40 colors. 0101 patterns are violet in even locations and green in odd locations, 1010 patterns are green in even locations and violet in odd locations, and 1111 patterns are white.

To select the 7 MHz LORES mode, select LORES-40 GRAPHICS and bring AN3 low (assumes 64K RAM card installed). This mode will also be selected if you bring up a LORES40 display and press and hold CONTROL-RESET. This brings RESET' low which brings AN3 low to force 7 MHz processing of LORES patterns.

It is very interesting that you can inhibit HIRES-40 delay processing by bringing AN3 low. This provides a means of changing the color of the entire



NOTE: [1] VID7M is low in double res timing.

Figure 8.7 TEXT Video Output (Left Side of Display).

Table 8.9 VID7M and LDPS' Timing HAL Logic Equations.

SIGNAL	EQUATIONS	NOTES
VID7M	S1 = GR'' • SEGB S2 = GR' • 80COL'' S3 = GR' • 7M S4 = VID7' • Φ1 • Q3' • AX' S5 = H0' • CLR REF • Φ1 • Q3' • AX' T1 = VID7M • AX T2 = VID7M • Φ0 T3 = VID7M • Q3	LORES GRAPHICS IS HIGH SPEED DOUBLE RES IS HIGH SPEED SAME AS 7M IF NOT HIRES FRCTXT'' • 80COL' ⟶ 7M, UNDELAYED HIRES DELAY CHECK AT Φ1 • Q3' • AX' NO DELAY AT RIGHT DISPLAY EDGE TOGGLE THROUGH AX KEEP TOGGLING THROUGH Φ0 KEEP TOGGLING THROUGH Q3
LDPS'	S1 = Q3' • AX' • 80COL'' • GR' S2 = Q3' • AX' • Φ1 • GR' S3 = Q3' • AX' • Φ1 • SEGB S4 = Q3' • AX' • Φ1 • VID7' S5 = Q3' • AX' • Φ1 • CLR REF • H0' S6 = Q3' • AX • RAS'' • Φ1 • VID7 • SEGB' • GR''	DOUBLE RES CAUSES DOUBLE LDPS' TEXT MODE LORES NOT DELAYED HIRES RIGHT DISPLAY EDGE CUTOFF HIRES DELAYED LDPS'

HIRES40 screen instantaneously, no small feat when you consider that you normally would have to look up addresses and change 7-dot patterns one by one to change the color of a HIRES object. To utilize this capability, store all HIRES patterns with D7 set. Then with 80COL reset and a 64K auxiliary RAM card installed (or a jumper between pins 50 and 55 on the auxiliary slot), you can switch between green/violet and orange/blue coloring via AN3. As a corollary, you can instantly shift the position of a HIRES40 display left or right 1/560 the width of the Apple display via AN3.

Instantaneous switching of HIRES colors suggests a use for the 7M LORES mode. You can create a display of LORES $5 and LORES $A blocks and instantaneously switch the entire LORES screen between gray/gray and green/violet coloring. I know you're racing to your desk to write home to Mom about this one.

The right side cutoff of HIRES40 delayed patterns referred to in item 7 above represents an improvement over the Apple II in which the rightmost dot is cut off or not, depending on the MSB of the first byte scanned during the following HBL. It would be preferable if delayed patterns were not cut off at the right side, but that would not have been particularly convenient to do in the Apple IIe timing hardware structure (they would have had to add a chip).

Figure 3.2 illustrates the right side cutoff of a HIRES40 delayed pattern along with other HIRES-40 variations of VID7M and LDPS'. Note the undelayed timing on the last video cycle in Figure 3.2, even though VID7 is high. Note also the double width LDPS' pulse, which occurs only at this forced cutoff point.

TEXT OUTPUT

Apple IIe TEXT output timing is very straightforward. The video scanner drives character code from RAM to the video data bus where it addresses the video ROM. SEGA, SEGB, and SEGC are equivalent to VA, VB, and VC, and these three address inputs to the video ROM select among the eight 7-dot segments of the character indicated by the code on the video data bus. One segment of a character pattern is thus output for each of the eight times a character code is driven out of RAM. The video ROM text patterns are illustrated in Figure 8.8.

Figure 8.7 shows the output of the seventh (out of eight) dot patterns of a NORMAL ampersand and an INVERSE ampersand to the first two character positions at the left of the screen. At the far left side of Figure 8.7, H5 through H0 of the video scanner are at 011000 meaning HBL has just gone low and WNDW' will go low on the next scanner clock (RAS' rising during PHASE 1). The video data bus contains inconsequential data from HBL scanned memory, and the PICTURE signal is blanked because WNDW' is high, causing a high level to be shifted continuously to PICTURE'.

NOTES: [1] "Mouse text" replaces letters and characters at ASCII = $40—$5F in the enhanced firmware video ROM (see Chapter 6, **The Apple IIe Firmware Upgrade**).

Figure 8.8 Apple IIe Video ROM Text Patterns.

The first time LDPS' falls at the left side of Figure 8.7, WNDW' is still high, and the video ROM output is still held high by the pull-up resistors. Consequently, ONEs are loaded into the load/shift register and the PICTURE signal remains blank until LDPS' falls again.

At about the same time that WNDW' falls, data from the H5—H0 = 011000 access to RAM is latched in the auxiliary card and motherboard video latches. The auxiliary video data then becomes valid on the video data bus about 20 nanoseconds after PHASE 0 rises. This data is not used unless 80COL is set, causing LDPS' to fall near the end of PHASE 0. Assume for now that 80COL is set and that the Apple is in TEXT80 mode. This is the timing shown at the bottom of Figure 8.7.

The video bus now contains $A6, ASCII for a NORMAL ampersand. The IOU translates VID7—6 = 10 to RA10—9 = 10, so the processed video data bus input to the video ROM is also $A6. SEGC-SEGB-SEGA = 110 because this is the seventh of eight segments, and GR+2 is low because TEXT mode is selected. The video ROM address is therefore GR+2, RA10—9, VID5—0, SEGC—A = 0,10, 100110,110. At this address, the video ROM contains 11010011, the inverted pattern of the seventh segment of a NORMAL ampersand.*

About 390 nanoseconds after the auxiliary video data becomes valid on the video data bus, 14M rises with LDPS' and VID7M low, loading the dot pattern to the load/shift register. The access time of the video ROM in the Apple IIe must therefore be under 390 nanoseconds. One would think that this would dictate the use of a 350-nanosecond video ROM, but don't bet on it. The Apple II requires the same 390-nanosecond access time for its text ROM, and 450-nanosecond text ROMs are used in the Apple II. I don't know if they decided to use a 350-nanosecond video ROM in the Apple IIe because the ROM is only labeled with Apple's part number, not the more descriptive manufacturer's part number. In any case I recommend using a 350-nanosecond part if you decide to install a custom video EPROM in your Apple IIe.

The 11010011 pattern represents only seven dot positions because only six shifts will take place before the next pattern is loaded. Only 1010011 will be shifted out, LSB first. The leading and trailing ONEs are present on all of the NORMAL text patterns in the video ROM and provide the necessary horizontal spacing between characters.

The first time LDPS' falls after WNDW' drops low marks the beginning of the display on the screen. This is also true of left side video output of LORES, HIRES undelayed, and HIRES delayed timing. Whatever the display mode, the left side blanking is extended until the first display pattern is loaded.

The 1010011 pattern is shifted to PICTURE', with one shift every 14M rising since VID7M is held low during TEXT time with 80COL set. Motherboard video data becomes valid about 20 nanoseconds after PHASE 0 falls, while the auxiliary pattern is being shifted out. The video ROM then has about 390 nanoseconds to respond by making the new pattern available for loading.

In the Figure 8.7 example, the auxiliary data on the video data bus is now $26, code for an INVERSE ampersand. VID7—VID6 = 00 is translated in the IOU to RA10—RA9 = 00, so the GR+2, RA10—9, VID5—0, SEGC—A video ROM address is 0,00, 100110,110. At this location, the video ROM contains 00101100, the dot pattern of the seventh of eight segments of an INVERSE ampersand, and the complement of the pattern generated by the NORMAL ampersand. The 00101100 pattern is loaded when LDPS' falls near the end of PHASE 1, and the resulting 7-dot pattern, 0101100, is shifted to the PICTURE' line, LSB first.

The code for the NORMAL and INVERSE ampersand in the DOUBLE-RES example are stored at the same address with the NORMAL code stored in auxiliary card RAM and the INVERSE code stored in motherboard RAM. This is an essential feature of all Apple IIe DOUBLE-RES processing—two video cycles per video scanner state. On the other hand, SINGLE-RES processing consists of one video cycle per video scanner state. This is implemented simply by loading video patterns only once per scanner state (when motherboard data is on the video bus) and shifting the pattern every other 14M rising (one half as fast as DOUBLE-RES shifting).

The NORMAL and INVERSE ampersand code in the SINGLE-RES example are stored in adjacent locations of motherboard RAM (H5—0 = 011000 and 011001). LDPS' doesn't fall during PHASE 0, and the pattern resulting from auxiliary video data is thus ignored. The blanking period at the left side is extended ½ microsecond beyond that of DOUBLE-RES processing, until the pattern resulting from motherboard data is loaded. The patterns loaded are identical to those of the DOUBLE-RES example, a NORMAL ampersand followed by an INVERSE

*The pattern is inverted here because the QH output of the load/shift register is connected to the PICTURE' line which is inverted and ORed with ALTVID' to produce the PICTURE signal. Therefore, all patterns in the video ROM are inverted (ONE = black; ZERO = white).

ampersand. Only the processing speed and consequent character horizontal size are different.

Notice the long delay, in Figure 8.7, between a video scanner state and the LDPS' which loads patterns resulting from code stored at a RAM location addressed by that video scanner state. This is why WNDW' and SEGA—SEGC are delayed by one scanner clock. Because of the delay, WNDW' stays high long enough that blanking ONEs are loaded and shifted until the time a displayed dot pattern is ready to be loaded and shifted out. Also, the delay causes SEGA—SEGC to address the video ROM simultaneously with video data resulting from the video scanner state which produced SEGA—SEGC.

LORES GRAPHICS OUTPUT

LORES blocks are not generated the way you would expect from looking at a television. You would expect big 1-microsecond pulses on the PICTURE signal would be required to produce those big one microsecond wide blocks. In reality, the only pulses that are one microsecond wide in LORES are white blocks. The colored blocks are made up of a string of narrow pulses, too narrow for a television to paint without blurring them into blocks, and narrow enough that they will be passed by the television's chrominance amplifier.

The LORES colored PICTURE signal swings back and forth between the black and white levels at 3.58 MHz. Color variations result from variations in the phase relationship between the PICTURE signal and COLOR REFERENCE. Also, the LORES 3.58 MHz PICTURE signal may or may not be symmetrical, and this adds greater variety to the available colors. All told, there are 12 LORES patterns which produce colored signals: 0001, 0010, 0011, 0100, 0110, 0111, 1000, 1001, 1011, 1100, 1101, and 1110.

The GR+2, RA10—9, VID5—0, SEGC, SEGB, SEGA address to the video ROM in LORES GRAPHICS mode is equivalent to 1, VID7—6, VID5—0, VC, 1, H0. GR+2 = 1 and SEGB = 1 together identify the LORES GRAPHICS area of the video ROM. VC and H0 divide the video ROM into four areas to implement certain details of LORES processing.

The contents of the LORES area of the video ROM are inverted double patterns, generated from VID3—VID0 or VID7—VID4 depending on VC; rotated or not rotated two bits depending on H0. The VC variation results in VID3—0 being processed as the upper block pattern and VID7—4 being processed as the lower block pattern. The H0 variation causes a stored pattern to generate the same color whether it is stored in an even or odd RAM location.

Table 8.10 shows the outputs of the video ROM corresponding to the stored byte, $27, as an example of the VC/H0 variations. When VC is low, LORES upper blocks are displayed and the lower four bits (0111) generate the pattern to be shifted. In even processing (as identified by H0'), the simple inversion of the 0111 pattern (1000) is output in the upper four bits and lower four bits of the video ROM. Rotating this 8-bit double pattern is the same as rotating the 4-bit pattern, so even processing of 0111 consists of rotating 1000.

When 0111 is driven from an odd RAM address, 0010 0010 is produced at the video ROM output. 0010 0010 is 1000 1000 rotated two bits (right or left—the result is the same). This rotation offset compensates for the fact that there are 3.5 cycles of COLOR REFERENCE in a SINGLE-RES video cycle. With a pattern pre-rotated by two dot widths, a LORES pattern is generated in a constant phase relationship with COLOR REFERENCE, whether it is stored in an even or an odd RAM location.

Table 8.10 also shows the lower block processing of the $27 example. This is identical to the upper block processing, except that 0010 is used to produce the even inverted pattern 1101 1101 or the odd inverted and rotated pattern 0111 0111 at the output of the video ROM.

Figure 8.9 (color section) shows several timing examples of LORES blocks—three in SINGLE-RES and two in DOUBLE-RES. These examples illustrate the nature of LORES timing. The examples shown could be VC' or VC processing because there is no difference between VC' and VC once the double pattern is loaded in the load/shift register.

Table 8.10 H0/VC Variations of LORES, VID7—0 = $27.

STORED PATTERNS	VC	H0	VIDEO ROM OUTPUT
$27 (0010 0111)	0	0	$88 (1000 1000)
$27 (0010 0111)	0	1	$22 (0010 0010)
$27 (0010 0111)	1	0	$DD (1101 1101)
$27 (0010 0111)	1	1	$77 (0111 0111)

LORES40 Output

The first three examples in Figure 8.9 show even/odd pairs of LORES40 blocks. This mode is selected by resetting 80COL, TEXT, and HIRES and setting AN3 (FRCTXT').* Gated GR+2' high and SEGB (LORES) forces VID7M low, so all LORES pattern circulation is at 14 MHz. LDPS' falls only during PHASE 1, so only the motherboard double pattern is loaded. The double pattern is shifted out 1.75 times (14 cycles/8 bits) before the next double pattern is loaded, so the 4-dot patterns are effectively shifted out 3.5 times to generate the 1-microsecond LORES block width. The 4-dot patterns are therefore output at 3.5 million patterns per second, not coincidentally, the frequency of COLOR REFERENCE. As a result, all LORES patterns except 0000, 0101, 1010, and 1111 cause the PICTURE signal to alternate at the COLOR REFERENCE frequency and produce colored blocks on the television screen.

The first example of Figure 8.9 shows the output produced by 1001 stored in an adjacent even/odd pair of memory locations. In the even cycle, 0110 0110 is loaded to the load/shift register, rotated to the PICTURE' line beginning with the LSB, and inverted and applied to the PICTURE signal (10011001100110). In the odd cycle, 1001 1001 is loaded and rotated to the PICTURE signal with inversion (01100110011001). In either an even or an odd cycle, the PICTURE signal is a symmetrical square wave with the same relationship to COLOR REFERENCE as HIRES orange.

The coloring of the left and right edge of a LORES block depends on the patterns of the adjacent blocks. If adjacent blocks are the same pattern, the PICTURE signal is continuous, meaning orange mates to orange with no off color fringe between the two blocks. Different colors mate together with a joining pattern which is not the same as either of the joining colors, creating a color fringe which is more or less prominent depending on the colors and whether they meet at an odd-even or even-odd junction. Even though the right and left side of the first Figure 8.9 example are colored orange, coloring here depends on the adjacent patterns.

The second example in Figure 8.9 is even 0111, light blue, followed by odd 1000, dark brown. These patterns produce asymmetrical 3.58 MHz square waves whose 3.58 MHz sinusoidal component is passed by the television's chrominance amplifier to produce different colors. The asymmetrical square waves are produced by patterns with only one bit set or only one bit reset. Those with only one bit set produce dark colors, because the PICTURE signal spends most of its time in the black. Conversely, the patterns with only one bit reset produce bright colors. As the example shows, the picture pattern at the border between colors 0111 and 1000 is a combination of the two separate patterns. Even 0111 followed by odd 1000 produces a violet border.

The light blue block shows that some things are predictable about bordering colors in LORES blocks. Any odd pattern which ends in the white level will combine with the left side of even 0111 to form a white border. The bright colors are particularly prone to forming white borders, because they're only one black period away from being white themselves.

The third example in Figure 8.9 is even 0101 followed by odd 1010. These are the two gray LORES patterns. They are gray, because the PICTURE signal they produce is 7 MHz, which will not be passed by the television's chrominance amplifier. Now gray is really white in a dark disguise. White light can come in many intensities as evidenced by a black and white television picture, and LORES patterns 0101 and 1010 are just less intense white. They are equal to each other in intensity, and are therefore identical shades of gray. This is why the technical overview stated there were 15, not 16, LORES colors including black and white.

Even though the two grays produce the same medium intensity colorless blocks, they are 180 degrees out of phase with each other. Thus, when 1010 follows 0101 there is a discontinuity in the waveform at the border between them and a resulting color fringe. This can be done on purpose to separate two gray solids horizontally, or it can be avoided by using only 0101 or 1010 in a display. A good practice would be to choose one gray over the other to minimize unpleasant fringe borders with other colors.

When the LORES colors are displayed side by side in numerical sequence, there is no apparent color continuity between them. The fact is that they form a circular pattern of eight color tones determined by the phase relationship to COLOR REFERENCE. This is not apparent when they are in numerical sequence, because video processing treats the 4-bit color data as a dot pattern, not a numerical value.

*If FRCTXT' is brought low and 80COL is not set, VID7M will alternate and the abnormal LORES mode described in the **VIDEO GENERATION TIMING SIGNALS** section will result. Detailed timing descriptions of the abnormal LORES mode are not presented in this book.

Figure 8.10 shows the PICTURE signals, compared to COLOR REFERENCE, which are produced by the various LORES patterns. The colors are shown in an order in which the picture pulse shifts right as the colors progress from top to bottom. A very interesting point becomes evident when looking at this figure. There are four color tone pairs: dark and light magenta, dark and light blue, dark and light blue-green, and dark and light brown. For example, the dark magenta pulse is surrounded equally on both sides by the light magenta pulse, and the horizontal center of both pulses is at the same point on the COLOR REFERENCE. As a result, they produce the same color tone, but a series of wide pulses is brighter and whiter than the series of narrow pulses with the same color tone.

Now color 0001 is usually referred to as magenta, and color 1011 is usually referred to as pink. This book has been calling pink "light magenta" to make the point that color 1011 looks like color 0001 with a lot of whiteness in it. Anyone who wants to is encouraged to call pink "pink."

Figure 8.11 (color section) is a photograph of the LORES colors based on their circular nature. This display was generated by constructing the LORES colors in HIRES80 mode. As is obvious from this photo, the same colors are available in HIRES80 that are available in LORES. HIRES80 was chosen to illustrate the circular nature of these colors because circles look better in HIRES than in LORES.

In this display, the different color tones are in different sectors of a circle and brightness is represented radially in the circle with dark at the center and white at the outside. Black, gray, and white cover all sectors of the circle, because they have no coloring. Black is the darkest color. Then comes 0001, 0010, 0100, and 1000. The grays, 0101 and 1010, are the same brightness as the HIRES equivalents, 0011, 0110, 1100, and 1001. Next brightest are 0111, 1110, 1101, and 1011. Brightest of all is white. Looking at the colors in this way should give you insights when you are trying to produce pleasing LORES or HIRES80 displays.

LORES80 Output

The last two examples in Figure 8.9 show the output of five adjacent LORES80 blocks. This mode is selected by resetting TEXT, HIRES, and AN3 (FRCTXT') and setting 80COL. This causes no change in speed of pattern shifting from LORES40. LORES40 shifting is already at 14 MHz, and nothing happens faster than 14 MHz in an Apple IIe. The only difference between LORES40 and LORES80 timing is that LDPS' falls during PHASE 0 in LORES80 as well as during PHASE 1, so that the auxiliary pattern is loaded and shifted out in addition to the motherboard pattern. The LORES80 patterns cut each other off after 6 shifts so 1.75 4-dot patterns are output per .5 microsecond video cycle.

Since patterns are shifted at the same rate in LORES40 as they are in LORES80, the same colors are available. In LORES80, however, the blocks are only half as wide. Additionally, a pattern stored in auxiliary card RAM produces a different color than the same pattern stored in motherboard RAM. This can be seen from the first LORES80 example of Figure 8.9, which shows what happens when the 0010 pattern is stored continuously in auxiliary card and motherboard display memory.

When 0010 is driven out of an even auxiliary card RAM location (left side Figure 8.9), 1101 1101 is loaded, shifted out LSB first, and inverted (0100 010). When this is followed by 0010 from an even motherboard RAM location, the identical 0100 010 PICTURE signal results, but this is not a continuation of the 4-dot auxiliary PICTURE signal where it was cut off. The result is that the color for the period spanning these two video cycles is blurred and unpleasant. This discontinuity exists throughout the example.

The PICTURE signal required to continue a 0100 010 chain is 0 0100 01. This chain is produced by 0100 stored in a motherboard RAM location, and the final example of Figure 8.9 shows that 0010 stored continuously in auxiliary card RAM combined with 0100 stored continuously in motherboard RAM produces a continuous display the same color as LORES-40 pattern 0100. A correlation exists here that can be extended to all the LORES patterns. To reproduce a motherboard color, store the motherboard color, rotated right one bit, in auxiliary card RAM. Table 8.11 illustrates this correlation for all the LORES patterns.

One shortcoming of LORES80 graphics is that the horizontal block size variation is greater than that of LORES40. Variation occurs in the size of a single block depending how much black space there is at the left and right side of the block. This variation becomes more pronounced with colors with fewer dots set, and it becomes particularly pronounced in LORES80 because the maximum block size is only seven dot widths instead of 14. As an extreme example, even 1000 or odd 0010 will cause the display of blocks only one dot wide in LORES80 GRAPHICS mode.

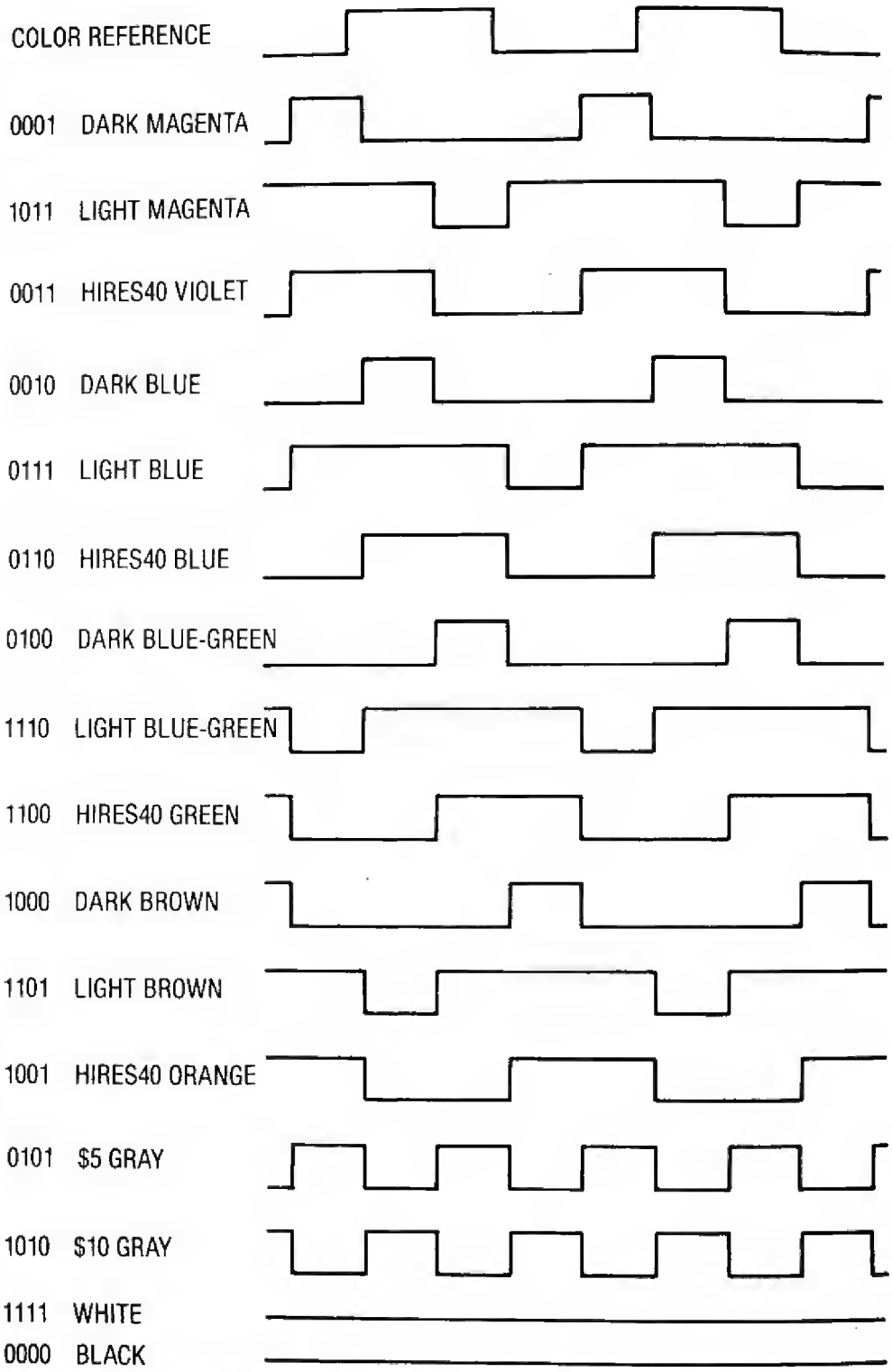


Figure 8.10 LORES/HIRES80 Patterns (C5—6) Compared to COLOR REFERENCE (C1—10).

Table 8.11 Auxiliary Card/Motherboard LORES Color Equivalents.

SINGLE RES COLOR	AUXILIARY CARD	MOTHERBOARD
$0 (0000)	$0 (0000)	$0 (0000)
$1 (0001)	$8 (1000)	$1 (0001)
$2 (0010)	$1 (0001)	$2 (0010)
$3 (0011)	$9 (1001)	$3 (0011)
$4 (0100)	$2 (0010)	$4 (0100)
$5 (0101)	$A (1010)	$5 (0101)
$6 (0110)	$3 (0011)	$6 (0110)
$7 (0111)	$B (1011)	$7 (0111)
$8 (1000)	$4 (0100)	$8 (1000)
$9 (1001)	$C (1100)	$9 (1001)
$A (1010)	$5 (0101)	$A (1010)
$B (1011)	$D (1101)	$B (1011)
$C (1100)	$6 (0110)	$C (1100)
$D (1101)	$E (1110)	$D (1101)
$E (1110)	$7 (0111)	$E (1110)
$F (1111)	$F (1111)	$F (1111)

LORES80 plotting is not supported by Apple IIe firmware. However, it is not a difficult matter to write Applesoft or Integer BASIC subroutines which utilize the LORES80 mode to generate 80 x 48 LORES block displays. Figure 8.12 (color section) is such a display, created by the Applesoft program of Figure 8.20 in an application note at the back of this chapter. As Figure 8.12 shows, LORES80 plotting gives very reasonable resolution considering the speed with which a display can be updated.

HIRES GRAPHICS OUTPUT

HIRES output has similarities to TEXT output. Both the HIRES and TEXT PICTURE signals are generated by shifting out 7-bit dot patterns. For this reason, it is possible to draw text using HIRES graphics with the same 7 x 8 dot patterns generated by the video ROM. The HIRES TEXT will have coloring however, because the COLOR BURST is enabled. Besides the COLOR BURST, important differences are:

1. HIRES patterns are stored directly in RAM, seven dots per byte. TEXT ASCII is stored in RAM, one character per byte, and TEXT patterns are stored in the video ROM, eight rows of seven dots per ASCII code.
2. HIRES40 7-dot patterns are delayed by one 14M period if the MSB (VID7) of the memory representation of the pattern is set and FRCTXT' is high.

While HIRES dot generation is a simple translation of dot patterns stored in memory to dot patterns on the screen, the coloring of the patterns is anything but simple. Figure 8.13 (color section) shows a number of color variations of the single dot pattern, 1011010. As these examples are discussed, it will be seen that HIRES coloration varies with single or double resolution, even or odd RAM location of the dot pattern, auxiliary or motherboard RAM location (HIRES80), delayed or undelayed timing of the displayed pattern (HIRES40), and delayed or undelayed timing of the adjacent displayed pattern (HIRES40).

The GR+2, RA10—9, VID5—0, SEGC, SEGB, SEGA address to the video ROM in HIRES GRAPHICS mode is equivalent to 1, VID7—6, VID5—0, VC, 0, H0. GR+2 = 1 and SEGB = 0 together identify the HIRES GRAPHICS area of the video ROM. The VID7, VC, and H0 address inputs do nothing but divide the ROM into eight identical areas. These addressing inputs affect LORES patterns, not HIRES patterns. In other words, **the HIRES GRAPHICS video ROM address is effectively VID6—VID0.**

The contents of the HIRES area of the video ROM is an inverted map of the VID6—VID0 address with the extraneous MSB set in all locations.* By this it is meant that the contents of VID6—0 = 0000000 is 11111111, the contents of VID6—0 = 0000001 is 11111110, etc. As a result, **the net effect of the video ROM in HIRES GRAPHICS is to invert the dot pattern on VID6—VID0.** The display map pattern in all of the examples of Figure 8.13 is 1011010, which produces the pattern 10100101 at the output of the video ROM.

*The MSB of the pattern from the video ROM is never shifted out in HIRES GRAPHICS, so it is extraneous. Apple had to put something in there, and chose to make it a ONE.

HIRES40 Output

The top two examples of Figure 8.13 show the four basic phase relationships between HIRES40 patterns and COLOR REFERENCE. The result is the same picture pattern colored four different ways. The four different colorings are produced by storing the pattern 1011010 at adjacent memory locations with D7 reset and at adjacent memory locations with D7 set.

The top HIRES pattern in Figure 8.13 is formed by 01011010 being driven out of RAM by scanner access to an even address followed by an odd address. Because the MSB is reset, the inverted (0100101) pattern will be loaded and shifted out undelayed.

To determine whether timing is to be delayed or undelayed for a HIRES pattern, VID7 is monitored at PHASE 1 • AX′ • Q3′ (marked with a dot in Figure 8.13) in the timing HAL. If VID7 is low at this point, as in the first example of Figure 8.13, LDPS′ and VID7M both fall on the following 14M rising. VID7M toggles on 14M rising at all other times, so undelayed VID7M is identical to 7M. The net result is that undelayed HIRES40 loading and shifting is identical to TEXT40 loading and shifting.

As the leading 101 is shifted out in the top example of Figure 8.13, the PICTURE signal starts black, swings white, then swings black, creating a square wave identical in frequency to the COLOR REFERENCE, 3.58 MHz. The television will pass this signal through its chrominance amplifier and phase compare it to the COLOR REFERENCE which it has reconstructed from the COLOR BURST. The result of this phase comparison will be color signals resulting in a HIRES green coloring of the dot on the screen. Compare the green dot position to the COLOR REFERENCE. Any PICTURE signal which goes white then black in this relationship with the COLOR REFERENCE will produce green coloring on the television.

Shifting along, we come to the two adjacent white dots. These dots are produced by a signal that goes white then black at 1.79 MHz, one half of the frequency of the COLOR REFERENCE. Very little of this signal can get through the television chrominance amplifier. The result is absence of color signals and subsequent white illumination. Anywhere on the screen, COLOR BURST or no COLOR BURST, bringing the PICTURE signal to the white level for a full period of COLOR REFERENCE will result in a white picture.

The white pulse is followed by a violet pulse, identical in pulse width to the green pulse but occurring in opposite phase relationship when compared to COLOR REFERENCE. HIRES green and HIRES violet complement each other; that is, they are 180 degrees out of phase.

When the identical 1011010 pattern is driven from an odd address to cause loading and shifting of 0100101, the PICTURE signal swings from white to black and back just as it did when the pattern was loaded from an even address. The COLOR REFERENCE, however, is 180 degrees out of phase from the way it was during the adjacent video cycle. The coloring of the screen dots is therefore the complement of the color of the pattern output in the adjacent video cycle. Green becomes violet, violet becomes green, and white remains white.

The nature of HIRES40 green and violet video should now be fairly clear. The HIRES40 dot is exactly the width of half of a COLOR REFERENCE period. Visualize the COLOR REFERENCE alternating up and down as the beam crosses the screen, starting with COLOR REFERENCE low as the display begins. If even position dots are turned on, they coincide with a quarter cycle of COLOR REFERENCE low followed by a quarter cycle of COLOR REFERENCE high, and are violet. Odd dots are in the opposite phase relationship with COLOR REFERENCE and are green, and two or more adjacent dots are white.

Many of the HIRES displays you see on the Apple IIe are made up of SINGLE-RES undelayed patterns. In fact, these were the only HIRES patterns that were available in the original Revision 0 Apple II. The undelayed HIRES40 mode gives you 280 programmable dots per scan with violet, green, or white coloring. If color is ignored, its resolution is 280 x 192 points, but if green or violet coloring is desired, only half of the horizontal dot positions can be used and the overall resolution is 140 x 192 points.

The output of the same 1011010 dot pattern, stored at adjacent memory locations with D7 set is shown in the second example of Figure 8.13. The picture pattern is the same, but it is delayed by one 14M period since VID7 is sampled high at PHASE 1 • AX′ • Q3′. With continuous delayed patterns as illustrated in this example, LDPS′ is low during the last 14M period of PHASE 1 and VID7M is equivalent to 7M′ (as opposed to equivalence with 7M).

Since the identical pattern is delayed by one 14M period from the previous example, a new pair of complementary colors, orange and blue, are produced. The new colors are a result of the fact that delayed dots have a different phase relationship with COLOR REFERENCE than undelayed dots. Along with the color change, the delayed pattern is shifted to the right one half of a HIRES40 dot width.

Delaying the 7-dot patterns is a tricky way of sprucing up the HIRES40 display (violet and green get a little old). Programming HIRES40 video becomes even more of an abstract art, however, with each group of seven dots having a color and position characteristic. Add this to the facts that alternating dot positions produce different colors, screen memory addresses are difficult to compute, and delayed HIRES patterns interfere with adjacent undelayed HIRES patterns; and you've got real programming complexity.

Delayed and undelayed HIRES patterns interfere with each other? They sure do, but before we get into that, let's summarize the characteristics of HIRES40 video based on the discussion to this point. First, there are 192 horizontal rows of dots. In each row, 280 dots (40 x 7) may be turned on or off, but since each group of seven dots may be shifted right half of a dot width, there are 560 dot positions in a row. Color depends on position, and there are 140 violet positions, 140 green positions, 140 blue positions, and 140 orange positions. Any two adjacent dots turned on will be white. We will see shortly that there are really only 139 orange positions. This is because an orange dot on the far right of the screen will be cut off by WNDW' to make it dark brown. Also, two adjacent delayed dots at the far right will be light blue-green, not white.

If color is of no concern, there is uninhibited programming of a 192 x 280 matrix of dots. With restrictions, this becomes 192 x 560. The main restriction is that a delayed dot cannot be in the same 7-bit group as an undelayed dot. For example, you can draw a slanted straight line close to the vertical with 192 x 560 resolution. You can also draw a very nice vertically oriented parabola with 560-dot horizontal resolution at the portions where the slope is more vertical. The other restriction on 560-dot resolution is the interference at the boundary between adjacent delayed and undelayed patterns which will be detailed shortly.

For coherent violet, green, blue, and orange colored displays, there is 192 x 140 dot resolution as long as certain pairs of colors don't get too close to each other. Anytime you plot a green dot in the same 7-dot pattern as an orange dot, that orange dot turns to green, because D7 had to be reset in that memory location to plot the green dot. Similar considerations exist for mixing blue and violet. Any two adjacent dots will always be white.

As was mentioned in the VIDEO GENERATION TIMING SIGNALS section, delayed timing is inhibited when FRCTXT' is low. The effect of pulling FRCTXT' low ($C05E) in Figure 8.13 would be to remove the delay from the PICTURE signal of the second example so that it is identical to the PICTURE signal of the first example. Orange is thus instantly changed to green, blue is changed to violet, and all delayed dots are shifted left 1/560 the width of the screen display.

Why do colored HIRES40 objects appear solid if every other dot is turned off? Shouldn't a violet object appear to be numerous horizontal rows of dots rather than solid lines? The object appears solid vertically because the horizontal scans are so close together. If you look at the violet object up close, you will see that it appears to be numerous horizontal lines. The reason that the object appears solid horizontally is that a multichannel color television is not capable of turning its beam intensity on and off cleanly at 3.58 MHz. Instead, the dots are blurred into continuous horizontal lines. For the same reason, Apple text is somewhat blurred when displayed on a television set.

Now if you inject the same VIDEO signal into a high frequency response video monitor, you will clearly see the black spots between the dots in the lines that were violet on the television. It is very educational to compare all forms of Apple video to simultaneous displays on a television and high frequency monitor. HIRES and LORES graphics modes use the "slowness" of a television to display colored solids, but the monitor shows the dot patterns which produce the solids. The "slowness" of a television is why computers that are designed to output TEXT to a television have a display of 40 TEXT characters or less. It is also for this reason that when you use the 80-column text display capability of the Apple IIe, you must support it with a high frequency response video monitor.

Interference Between Adjacent Delayed and Undelayed HIRES40 Patterns

The 7-dot, HIRES40 patterns fit snugly together if the adjacent patterns are all delayed or undelayed, but problems can be caused when they are mixed together. This can be seen from the third example of Figure 8.13 which shows the processing of the same 1011010 pattern stored in adjacent locations with D7 set in the even locations and reset in the odd locations.

An undelayed video shift-cycle is just being completed at the far left of example 3, so VID7M is initially in phase with 7M. Then a delayed cycle is initiated by the fact that VID7 is high at PHASE 1 • AX' • Q3'. This delays LDPS' and VID7M so that the next pattern is delayed—but think what it does to the present pattern. Since the loading of the new

pattern is delayed, the last dot of the previous cycle is extended by half of a dot width. In other words, **a delayed HIRES40 pattern extends the trailing dot or black space of a preceding undelayed pattern by half of a dot width.**

Note that VID7 is sampled for delayed timing only during PHASE 1, not during PHASE 0. This is because the HIRES delay only can occur in HIRES-40, not HIRES80 mode. The auxiliary card data is on the video data bus during PHASE 0, and this is not used in SINGLE-RES modes. Only the motherboard data on VID7 of the video data bus during PHASE 1 is sampled to determine delayed or undelayed timing.

The delayed pattern is shifted out until the decision point for delayed or undelayed timing on the following cycle is reached. VID7 is reset now, indicating that the following pattern is undelayed. Consequently, the timing HAL holds VID7M low and drops LDPS' on the next cycle to cause undelayed timing. The new pattern, however, is loaded when the last dot of the delayed pattern has only controlled the PICTURE' line for half of the normal period. In other words, an **undelayed HIRES pattern cuts off the trailing dot or black space of a preceding delayed pattern to half of a dot width.**

The point of all this is that continuous undelayed or delayed patterns fit snugly together, but there is discontinuity between adjacent undelayed and delayed patterns. Cutting off or extending a dot has the effect of slightly changing the dot pattern and, more noticeably, changing the coloring of the border dots. As a result, the HIRES programmer has one more thing that affects color to educate himself about and take into account. On the plus side, the programmer can draw vertical lines at pattern borders in eight colors that are not otherwise available in HIRES40 mode. He does this simply by turning on a right hand dot then extending or cutting it off via D7 of the following pattern. In some instances, no dots need be turned on in the following pattern.

Figure 8.14 (color section) is a photograph illustrating the generation of LORES colors at borders between delayed and undelayed 7-dot HIRES patterns. The program which generated this display is listed in Figure 3.11. The mixed LORES/HIRES display is created by switching screen modes in a 8515-cycle loop as is discussed in an application note in Chapter 3. As the photo shows, any LORES color except dark blue-green (4) and dark magenta (1) can be produced at a limited number of screen positions. LORES colors 3, 6, C, and 9 are natural equivalents of the HIRES40 colors. LORES colors 7 and 2 can be produced at even/odd memory addressing borders. Colors D and 8 can be produced at odd/even borders. Colors B and E can be produced at odd/even or even/odd borders. LORES color E can also be produced at the far right of the HIRES screen. Finally, orange HIRES dots at the right side of the screen are LORES dark brown (8).*

Figure 8.15 shows some patterns created by mixing delayed and undelayed dot patterns. Comparison to Figure 8.10 in the preceding LORES section will show that these patterns have the same phase relationship with COLOR REFERENCE as various LORES colors. The exception is the case where a trailing 1-0 delayed pattern is cut off by an undelayed 1-0 pattern. This creates green or violet with a different shade than any HIRES40 or LORES color. The general rule of all of these HIRES40 interference patterns is that delayed extends undelayed, and undelayed cuts off delayed.

Interference borders can be used to display isolated dots or vertical line segments in HIRES40 mode that are not one of the four HIRES40 colors. Mostly, though, interference borders are a nuisance. Anytime two different colors get close to each other horizontally, the video pattern at their border is different than either of the solid colors when compared to COLOR REFERENCE. Awareness of the causes of off color fringes should help you experiment with color combinations that produce eye pleasing displays.

The extreme left and right dot positions are special cases for delayed patterns. An undelayed pattern at the far left may extend the previous undelayed pattern, but this extended pattern will be the final blanked pattern of HBL. The result is that the delayed pattern at the far left pulls the black margin to itself.

A delayed pattern at the far right always has its final dot position cut in half. This is because of the special logic in the timing HAL that always forces the first HBL pattern to undelayed, regardless of VID7. On the right side of the screen, the last delayed dot is in an orange dot position. This dot is cut off by HBL so the dot is dark brown rather than orange. Similarly, a delayed white pair at the far right is cut off to make it light blue-green.

The handling of right and left side delayed patterns represents an improvement of the Apple IIe over the Apple II. In the Apple II, delayed patterns on the left side can inadvertantly extend undisplayed dots into the display, and the far right

*The references to LORES color numbers here are valid for LORES40 or LORES80 motherboard resident patterns. Auxiliary card resident LORES80 patterns result in different colors.

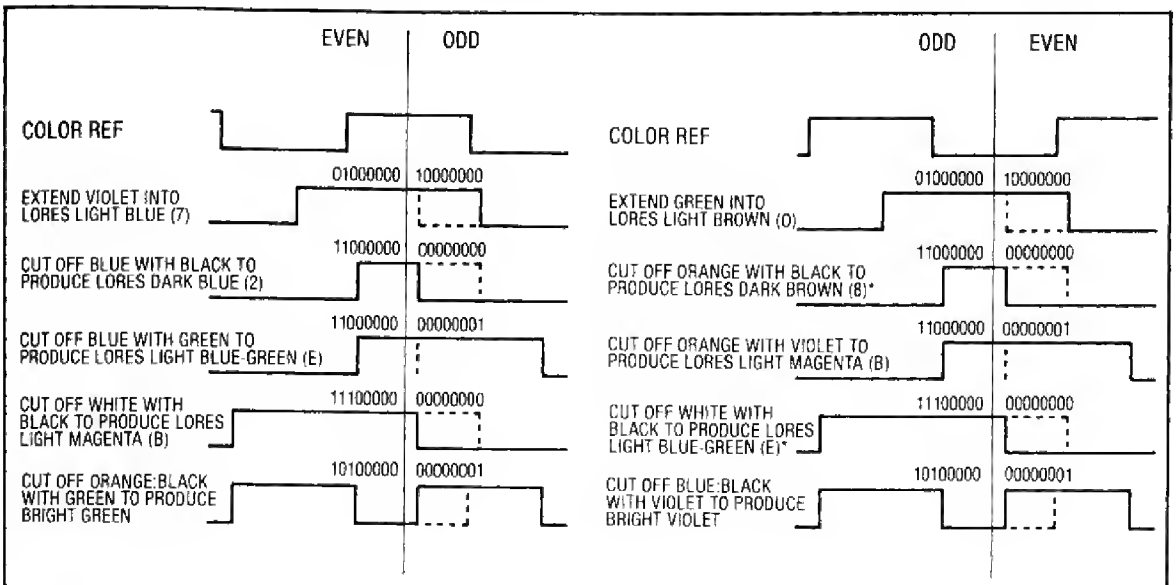


Figure 8.15 Timing Diagram: HIRES40 Interference at Delayed/Undelayed Pattern Borders.

delayed dot may or may not be cut off by undisplayed patterns. The Apple IIe is far more predictable, although it would have been nice if they had figured out a way to eliminate the right side cutoff of delayed patterns without extending right side undelayed patterns.

HIRES80 Output

The final example of Figure 8.13 shows our well used 1011010 pattern stored consecutively in auxiliary and motherboard RAM with the Apple IIe in HIRES80 mode. This mode is selected by resetting TEXT and AN3 (FRCTXT') and setting HIRES and 80COL. Like double resolution TEXT, double resolution HIRES is achieved by simply doubling the speed of the load/shift cycle so that the auxiliary video data is loaded and shifted out, alternating with motherboard video data. The delayed timing is inhibited when FRCTXT' is low, so HIRES80 LDPS' and VID7M timing is identical to that of TEXT80.

The PICTURE signal resulting from HIRES80 processing of the 1011010 pattern is identical to that resulting from HIRES40 processing, except it is compressed so that the dot positions are half as wide. On a high resolution monochrome monitor, this has the effect of displaying the same patterns compressed horizontally. On a television set or NTSC color video monitor, this has the effect of blurring the pattern and creating some washed out unattractive coloring. The 1011010 pattern is blurred because it is too fine to display on a television. The colors aren't any good because there is no coherent relationship with COLOR REFERENCE.

The fact that the picture is compressed and still clearly visible on a high resolution monitor represents the primary advantage of HIRES80 graphics. Instead of the 280 x 192 resolution of HIRES40 mode, the programmer has complete on/off control of a 560 x 192 matrix of dots. The improvement here is obvious, and not many words will be wasted here on the advantages of 560-point monochrome resolution over 280-point monochrome resolution. Rather, many words will be wasted describing the less obvious color features of the HIRES80 displays.

A complete cycle of COLOR REFERENCE lasts for two HIRES40 dot widths or four HIRES80 dot widths. As a result, alternating HIRES40 dots exactly match the COLOR REFERENCE frequency and consequently produce coherent colors. To achieve coherent colors in HIRES80 processing, the dot patterns must be set up so that they recur at 3.58 MHz. Storing the identical 7-dot pattern continuously as in Figure 8.13 cannot achieve this because the period of a seven dot video-cycle is not an integer multiple of the 3.58 MHz period.

Notice, in Figure 8.13, that a complete cycle of COLOR REFERENCE lasts exactly four HIRES80 dot widths. This is the key to producing coherent color in HIRES80 displays. If any 4-dot pattern other than 1111 or 0000 is shifted continuously to the PICTURE' line, the PICTURE signal will alternate up and down at 3.58 MHz and produce coherent color on the CRT. This is a burdensome programming task to be sure, but the resulting high resolution color displays can be very impressive.

The colors available via continuous 4-dot HIRES 80 patterns are the same colors that are available in LORES graphics (see Figures 8.10 and 8.11). In fact you can emulate LORES graphics from HIRES80 mode by causing 4-dot patterns to be continuously shifted out for 14 dot widths (3.5 COLOR REFERENCE periods) in four adjacent horizontal scans. The LORES emulation mode would have the advantages that you could shift blocks vertically by ¼ a block height or horizontally by 1/14 a block width, and that you could freely intermix emulated LORES blocks with HIRES points and lines. The disadvantage would be that considerably more computation time and memory space would be required to maintain the emulated LORES display.

With LORES graphics, you can shift any 4-dot pattern to the PICTURE signal for 14 dot widths. With HIRES80 graphics, you can shift any 4-dot pattern for any number of dot widths you desire. If you want to produce color blocks 8 dot widths by 2 horizontal scans, you can do it. Even if only one dot is shifted out, it will have coloration, although the color of a single dot isolated in a field of black is hard to discern.

A repeating 4-dot pattern is shifted 1.75 times in a HIRES80 video cycle. To continue shifting the same colored pattern in the next video cycle, the contents of the next scanned location of RAM must be the same as the current byte rotated left once. For example, if 0010 0010 is stored at an even auxiliary card location, its color can be matched by storing 0100 0100 in an even motherboard location, 1000 1000 in an odd auxiliary card location, or 0001 0001 in an odd motherboard location. Note that having the MSB set in the 1000 1000 stored byte has no effect on the display. It is set here only because it makes it easy to visualize and compute the rotation of 4-dot patterns.

The 4-dot patterns result in different colors depending on whether they are driven out of even or odd auxiliary card or motherboard RAM locations. When driven out of an even motherboard location, the pattern will result in the same color as it would in LORES40 mode. The HIRES80 patterns required to produce the various LORES colors are shown in Table 8.12.

There are any number of HIRES80 color plotting methods that can be used by Apple IIe programs. One method would be to utilize 560 x 192 plotting routines to selectively construct colored objects at various points on the screen. Another method would be to utilize super LORES routines which would

Table 8.12 HIRES80/LORES Color Equivalents.

	EQUIVALENT HIRES80 PATTERNS			
LORES COLOR	AUX/EVEN	MBD/EVEN	AUX/ODD	MBD/ODD
$0 (0000)	$00 (0000 0000)	$00 (0000 0000)	$00 (0000 0000)	$00 (0000 0000)
$1 (0001)	$88 (1000 0000)	$11 (0001 0001)	$22 (0010 0010)	$44 (0100 0100)
$2 (0010)	$11 (0001 0001)	$22 (0010 0010)	$44 (0100 0100)	$88 (1000 1000)
$3 (0011)	$99 (1001 1001)	$33 (0011 0011)	$66 (0110 0110)	$CC (1100 1100)
$4 (0100)	$22 (0010 0010)	$44 (0100 0100)	$88 (1000 1000)	$11 (0001 0001)
$5 (0101)	$AA (1010 1010)	$55 (0101 0101)	$AA (1010 1010)	$55 (0101 0101)
$6 (0110)	$33 (0011 0011)	$66 (0110 0110)	$CC (1100 1100)	$99 (1001 1001)
$7 (0111)	$BB (1011 1011)	$77 (0111 0111)	$EE (1110 1110)	$DD (1101 1101)
$8 (1000)	$44 (0100 0100)	$88 (1000 1000)	$11 (0001 0001)	$22 (0010 0010)
$9 (1001)	$CC (1100 1100)	$99 (1001 1001)	$33 (0011 0011)	$66 (0110 0110)
$A (1010)	$55 (0101 0101)	$AA (1010 1010)	$55 (0101 0101)	$AA (1010 1010)
$B (1011)	$DD (1101 1101)	$BB (1011 1011)	$77 (0111 0111)	$EE (1110 1110)
$C (1100)	$66 (0110 0110)	$CC (1100 1100)	$99 (1001 1001)	$33 (0011 0011)
$D (1101)	$EE (1110 1110)	$DD (1101 1101)	$BB (1011 1011)	$77 (0111 0111)
$E (1110)	$77 (0111 0111)	$EE (1110 1110)	$DD (1101 1101)	$BB (1011 1011)
$F (1111)	$FF (1111 1111)	$FF (1111 1111)	$FF (1111 1111)	$FF (1111 1111)

plot a block of any height (1—192) and any width (1—560) of any color (0—15) at any location (X,Y = 0—559,0—191).

A third method would be to utilize a 140 x 192 colored HIRES mode with each 1/140 position consisting of a blue (560 MOD 4 = 0), blue-green (560 MOD 4 = 1), brown (560 MOD 4 = 2), and magenta (560 MOD 4 = 3) dot. The color of each position is determined by which of its four dots is turned on. The left side of Figure 8.16 (color section) shows some sinewaves plotted in this format.* This method provides true 140 x 192 resolution in 16 colors (including black, white, 0101 gray and 1010 gray).

Notice the jagged slopes of the sinewaves on the left side of Figure 8.16. These can be smoothed out to some extent on all colors except the 1-dot colors (equivalents of LORES 1, 2, 4, and 8). The method involves plotting 557 x 192 4-dot HIRES positions. This takes more time and programming overhead than simple 140 x 192 plotting, but line smoothness approaches that of 560 x 192 resolution for the 3-dot colors and white (see the sinewaves on the right side of Figure 8.16). The smoothness of gray lines can be further improved if use of 0101 gray and 1010 gray is alternated every horizontal scan. The effective resolution of the 1-dot colors cannot be improved because only 140 dots per horizontal scan can be displayed.

The HIRES80 color programming methods described here are only ideas which seem very basic to me. These ideas could be expanded and mixed with other ideas to provide some general purpose machine language HIRES plotting routines which support all the graphics features of the Apple IIe. This is something that is very badly needed because only the HIRES40 features are supported by Apple IIe firmware.**

MIXED MODE SWITCHING

A final topic to consider in video generation is MIXED mode switching. This is the reason we have to live with those delayed GRAPHICS time terms (GR+1 and GR+2), so we'll have a closer look.

Figure 8.17 is a timing diagram of the last display cycle of line 159 in MIXED mode. At the left side of this figure, HPE and H5—H0 of the video scanner switch from 1111111 to 0000000, marking the beginning of HBL. At the same time the horizontal section of the scanner goes to zeroes, the vertical section goes to 110100000 making the term V4 • V2 true. This identifies TEXT time, but you can't immediately switch to TEXT processing because the final displayed GRAPHICS pattern is not yet loaded in the load/shift register. For that matter, you can't start blanking yet either.

What happens is that all GRAPHICS time switching is delayed by two scanner clocks (see GR, GR+1, and GR+2 in Figure 8.5). IOU outputs affected by this delay include SEGA, SEGB, RA9, RA10, and GR+2. Because of the 2-clock delay, the timing HAL and video ROM are configured for GRAPHICS until the last video cycle on the right side of the screen is complete and blanking ONEs are loaded in the load/shift register. Here is the order of events for Figure 8.17.

1. RAS' rises during PHASE 1, bringing the video scanner to 110100000/0000000 after propagation delay. V4 • V2 identifies TEXT time.
2. At about the same time the video scanner changes states, PHASE 0 rises, latching the final GRAPHICS data in the motherboard video latch.
3. LDPS' falls during PHASE 0 of DOUBLE-RES modes, and the final auxiliary pattern is loaded and shifted out.
4. LDPS' falls during PHASE 1, loading the final motherboard pattern. In DOUBLE-RES or SINGLE-RES modes, this pattern will be the last one displayed before the right blanking margin.
5. RAS' rises during PHASE 1, followed by GR+1 falling and WNDW' rising. GR+1 selects TEXT related terms for SEGA—SEGC (VA, VB, and VC), but this change will not be felt at the SEGA—SEGC lines until RAS' rises during PHASE 1 again. GRAPHICS time switching on SEGA—SEGC is thus delayed by two scanner clocks. GR+1 also disables HIRES time at the IOU RAM addressing circuitry so the TEXT/LORES area of scanned memory will now be addressed.
6. At about the same time GR+1 falls, PHASE 0 rises to latch the first undisplayed video data. This data will come from HIRES memory if HIRES is set since HIRES TIME is just now falling.
7. The final pattern is shifted out until LDPS' falls again, loading the blanking ONEs resulting from WNDW' high.

*Figure 8.16 was produced by an Applesoft program (Figure 8.21). An application note at the back of this chapter describes some Applesoft techniques for programming HIRES80 displays.

**For alternate perspectives of HIRES80 operation and an assembly language listing of some HIRES80 plotting routines, see "True Sixteen-Color Hi-Res" by Allen Watson III, *Apple Orchard*, January, 1984.

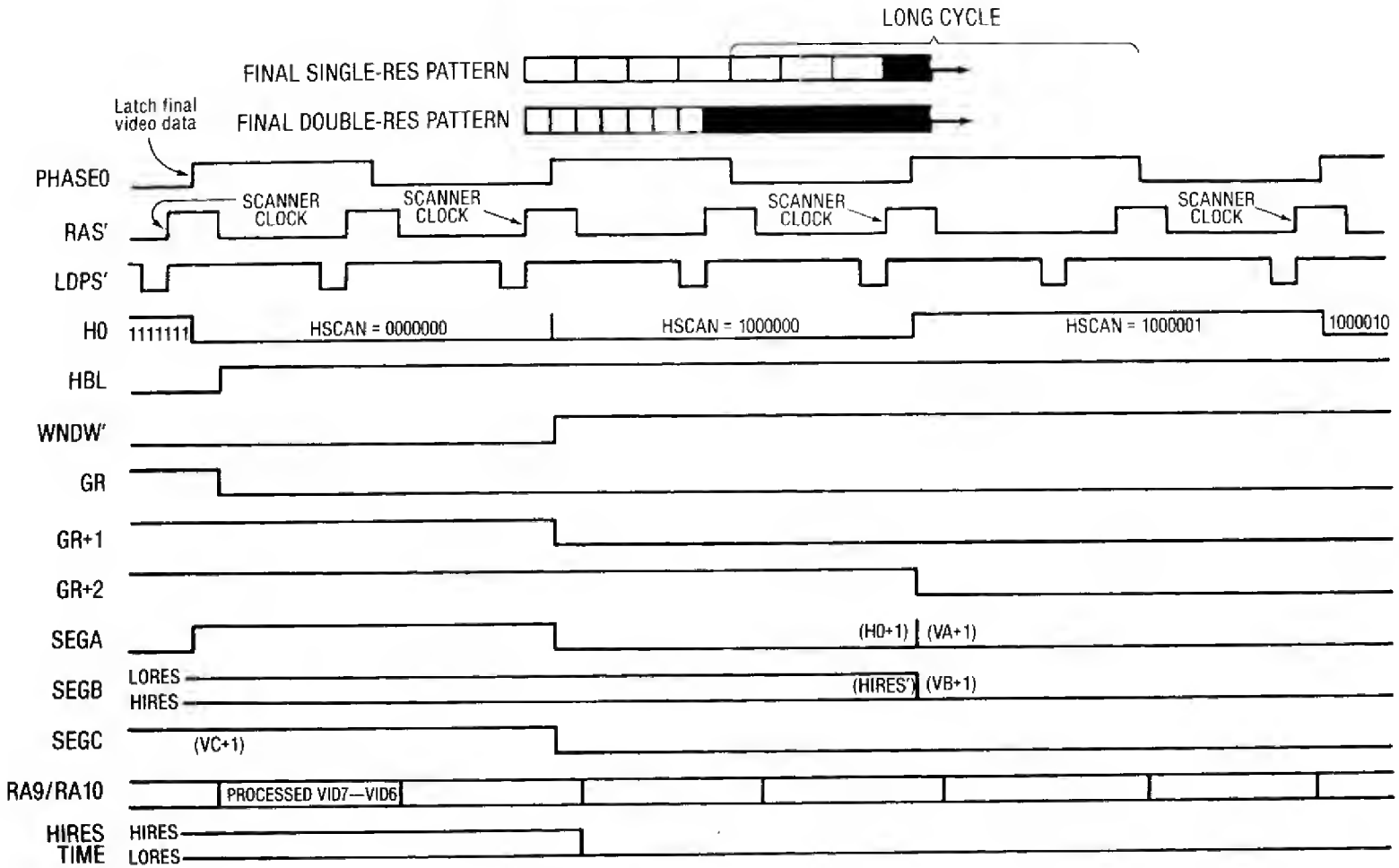


Figure 8.17 Switching from GRAPHICS to TEXT in MIXED Mode.

8. RAS' rises during PHASE 1, followed by GR+2. SEGA—SEGC, RA9—RA10, and GR+2 are now completely in TEXT configuration. Note that this does not occur until the last display pattern is completely shifted out.
9. At about the same time GR+2 falls, PHASE 0 rises to latch the second undisplayed video data. Even if HIRES is set, this data will come from TEXT/LORES memory since HIRES TIME has been low for about one microsecond.

As far as video ROM addressing is concerned, it would suffice to delay TEXT configuration only one scanner clock because the final pattern is loaded in the load/shift register after one clock. This would not suffice, however, for timing HAL inputs because LDPS' and VID7M must support GRAPHICS shifting until the last pattern is shifted and the blanking ONEs are loaded. Therefore, the 2-clock delay is necessary on SEGB and GR+2.

Ideal SINGLE-RES GRAPHICS/TEXT timing would have video ROM addressing inputs switching on GR+1 and timing HAL inputs switching on GR+2. This could be done in the Apple IIe if separate HIRES and GRAPHICS identifying terms were developed for the timing HAL. As it is, separate terms are not developed and all terms are switched on GR+2.* This creates a glitch in the screen when programs switch between GRAPHICS and TEXT during display time using the $C050/$C051 TEXT soft switch.

The switch back to GRAPHICS in MIXED mode occurs at the beginning of HBL before the first displayed horizontal scan, and the timing is similar to that of Figure 8.17. This timing is not particularly critical because the WNDW' is high during the entire transition period. Also, as was mentioned in Chapter 5, the display mode is switched to GRAPHICS and back to TEXT during VBL, but this is inconsequential because it happens during blanking time.

*HIRES TIME in the RAM addressing circuitry is switched at GR+1, and, because of RAM access delay, this switches the video data bus between HIRES and TEXT/LORES data in time alignment with the video ROM address inputs that are switched on GR+2.

PROGRAMMING SCREEN CHARACTER SETS IN EPROM

The video ROM of the American Apple IIe is pin compatible with 2732 4K EPROM. If you know someone with a PROM burner and understand the layout of the video ROM, you can customize the screen display of your Apple IIe by burning a custom video EPROM and installing it in place of the video ROM.

There are a number of aspects of the video display that it is possible to customize. You could change the HIRES patterns so the dots appear on the screen in the same order as they are represented in memory. You could change the LORES layout so that, as numerical color representations increase, the screen colors will sequence smoothly through the Apple spectrum (see Figure 8.11). Of course very few people would want to change their Apple this way, even if they preferred the resulting operational features, because most Apple graphics oriented programs would not work correctly.

One change you can make to the video ROM that has no undesirable side effects is to change the screen text character patterns. You may design your own upper case/lower case set or use an existing upper case/lower case set in place of the NORMAL set that resides in the video ROM. You may also design your own INVERSE set or FLASHING set which will be displayed anytime a program outputs inverse or flashing text, but they don't have to be inverted or flashing characters. This application note contains some suggestions for burning video EPROMS with custom text character sets. Please refer to Figure 8.8 and Tables 8.3 and 8.4 during these discussions.

The TEXT patterns are in the lower half of the video ROM ($000—$7FF). They are laid out identically to the ALTCHRSET ASCII (see bottom half of Table 8.4). In fact, you can compute the base addresses in the video ROM of any character by multiplying its ALTCHRSET ASCII representation by eight. For example, the ASCII for a NORMAL upper case "B" is $C2. Multiplying $C2 x 8 yields $610, and the eight segments of the NORMAL upper case "B" can be found in the video ROM at $610—$617.

The precise contents of $610—$617 in the video ROM are $E1 $DD $DD $E1 $DD $DD $E1 $FF. This makes much more sense if you look at the numbers in binary, stacked over each other as shown below. It becomes clear that the ZEROes in the numbers represent the dots in a matrix that form the letter "B". When you see a NORMAL "B" on the TEXT screen of the Apple IIe, it is a direct consequence of the fact that these numbers (patterns) are stored at $610—$617 of the video ROM.

```
11100001          ØØØØ
11011101        Ø    Ø
11011101        Ø    Ø
11100001    =     ØØØØ
11011101        Ø    Ø
11011101        Ø    Ø
11100001          ØØØØ
11111111
```

By looking at the "B" pattern, you can learn everything there is to know about the format of stored patterns in the video ROM. Without mincing words, they are reversed (mirror image), inverted (ZERO = dot) patterns stored in the seven LSBs. The extraneous MSB is set, but this bit is never shifted out so it doesn't matter whether it is set or reset. The patterns are bordered by blank spots on the left, right, and bottom to create spaces between characters on the screen, but you can vary this to implement the features of a character set.

One very simple way of personalizing your TEXT display is to change the pattern at $7F8—$7FF. This area corresponds to ASCII = $FF, and it contains a little checkerboard pattern in the video ROM. Apple IIe 40-column firmware generates a flashing cursor by periodically storing $FF at the memory location corresponding to the cursor position, so you can personalize the cursor by changing the pattern at $7F8—$7FF. For example, program an EPROM identical to the video ROM at all locations except $7F8—$7FF. At these addresses, replace $FF $D5 $EB $D5 $EB $D5 $FF $FF with $FF $C1 $DD $DD $DD $C1 $FF $FF and your cursor will be a little square instead of a checkerboard.

The TEXT area of the video ROM is divided up as shown in Table 8.13. The NORMAL special, upper, and lower area ($500—$7FF) contains the primary displayed character set of the Apple IIe. If you like, you can install a different character set here. Of course, this set should be very readable, especially if you do much text processing with your Apple. The idea is to personalize and possibly improve the text display, not to become an eye drop junkie.

Table 8.13
Text Pattern Addressing in the Video ROM.

ADDRESS	CHARACTER GROUP
$000—$0FF	INVERSE control (upper)
$100—$1FF	INVERSE special
$200—$2FF	INVERSE upper*
$300—$3FF	INVERSE lower
$400—$4FF	NORMAL control (upper)
$500—$5FF	NORMAL special
$600—$6FF	NORMAL upper
$700—$7FF	NORMAL lower

*The $200—$2FF INVERSE character patterns are replaced by mouse icon patterns in the enhanced firmware video ROM (see Figure 8.8).

Where can you get a full ASCII set of text patterns? There are numerous sources for character set patterns. One source is the *DOS TOOLKIT* distributed by Apple. Among other things, this valuable disk contains 21 character sets and *ANIMATRIX*, a program which implements computer aided design of other character sets. These HRCG (HIRES Character Generator) sets are meant to draw text on the HIRES screen, but they are identical in format (except for inversion) to the character sets in the video ROM. You can program the inversion of an HRCG set into a $500—$7FF video EPROM, and the HRCG set will become the primary displayed TEXT character set of your Apple IIe.

A second adaptation that must be made to HRCG character sets is to offset the different effect of bit 7 in the video ROM and in a HIRES40 pattern. While bit 7 in video ROM text patterns does nothing, bit 7 set in a HIRES40 pattern causes the pattern to be delayed half of a dot position. In the *DOS TOOLKIT* sets, D7 is occasionally set to improve the smoothness of a character. These characters would look a little cockeyed without the delay, so they need to be modified before using them in your video EPROM. The way to do this would be to load the set into *ANIMATRIX* where the few delayed patterns can be easily spotted and the character modified for symmetry with no pattern delays. Needless to say, if you use *ANIMATRIX* to design your EPROM text patterns from scratch, don't use the delay feature.

You can also customize INVERSE or FLASHING text on the Apple IIe, but some thought must be given to the effects of ALTCHRSET. With ALTCHRSET set, the $100—$3FF area is used as a full ASCII INVERSE set, but with ALTCHRSET set, $000—$1FF is used as an upper case only INVERSE set, and FLASHING text is created by switching between the $000—$1FF and $400—$5FF areas. This creates several options for interesting character sets.

One option is to change the entire INVERSE area ($000—$3FF) to an alternate set. This would not necessarily have to be an inverted set. Anything other than the NORMAL patterns (Italics for instance) would serve to highlight the display. FLASHING text would still flash, except flashing would be between standard and alternate character sets instead of between inverted and not-inverted sets.

Another option is to leave $100—$3FF alone but make the characters at $000—$0FF some alternate inverted set. This would signal the operator whether a program was using ALTCHRSET' or ALTCHRSET inverted upper case letters. The difference between the $000—$0FF and $200—$2FF characters need only be very slight if your intention is just to differentiate between ALTCHRSET' and ALTCHRSET upper case letters. A dot in a lower corner of the patterns would suffice.

Still another option is to place a graphics set or alternate alphabetic set at $400—$4FF. FLASHING text would still flash, but it would flash between the $000—$1FF characters and whatever you stored at $400—$4FF. Additionally, you would know anytime some control ASCII got into display memory, and, if you made this area a graphics set, you could program some pretty fancy text displays. Note that you would have to find another means than COUT1 to get the graphics code into display memory since COUT1 treats control code as control code, not display output.

If your Apple IIe has the firmware upgrade installed, some of the ground rules for modifying the video EPROM are different. The $200—$2FF INVERSE text patterns are replaced with mouse icon patterns in the new video ROM, and the 80-column firmware video output routine is changed so that ALTCHRSET INVERSE upper case output results in storage of $0—$1F in the display map instead of $40—$5F. With the "mouse text" video ROM, such options as deleting, modifying, and replacing icons are open to chronic Apple modifiers.

The video ROMs in Apple IIe's with PAL motherboards (European Apples) are equivalents of 2764s instead of 2732s. These video ROMs have a foreign language INVERSE/NORMAL screen character set in addition to the American characters. If the owner of a PAL-based Apple IIe does not need one of the video ROM sets, he can replace it with any INVERSE/NORMAL set he desires in a custom 2764 EPROM. Owners of NTSC based Apples can

obtain the same capability by building the adaptor shown in Figure 8.18. This adaptor allows installation of a 2764 video EPROM in the NTSC motherboard with mechanical switch selection between character sets.

The 2764 video EPROM adaptor could become very useful if the firmware upgrade is installed in your Apple IIe. Some of your software may not use firmware routines for video output and might not correctly output INVERSE video with the new video ROM. The solution to this problem is to adapt the video ROM socket to 2764, then program the contents of the old video ROM into one half and the new video ROM into the other half of a 2764. With this 2764 installed in the video ROM socket through the Figure 8.18 adaptor, you can switch between the icon set and the INVERSE set at will.

The best way to build your EPROM source file is to start with the video ROM, and just change the parts you wish to modify, leaving the GRAPHICS area intact. Assuming you only have one Apple with PROM burner available, the way to make a copy of the video ROM is to turn off the Apple, remove the video ROM, and install it in your PROM burner. Then turn the Apple on and enter PR#1 to enable your printer. With no video ROM, the screen will be blank, but you should be able to manipulate the Apple using the printer as a replacement for video output. Activate your PROM burner program, read the video ROM to RAM, and save it to a disk file. Then turn off the Apple and reinstall the video ROM.

If, for some reason, you cannot generate a disk file of the video ROM contents as described above, you can run the Applesoft program in Figure 8.19. This program generates the GRAPHICS portion of the video ROM and stores it in a disk file named VIDROM/2. Merge this file with some selected text character sets to complete your video EPROM source file. If you want to experiment, rearrange the DATA statement of line 150. This DATA statement contains the decimal equivalents of the LORES double patterns ($00, $11, $22, etc.) for H0'. Rearranging them will rearrange the LORES colors

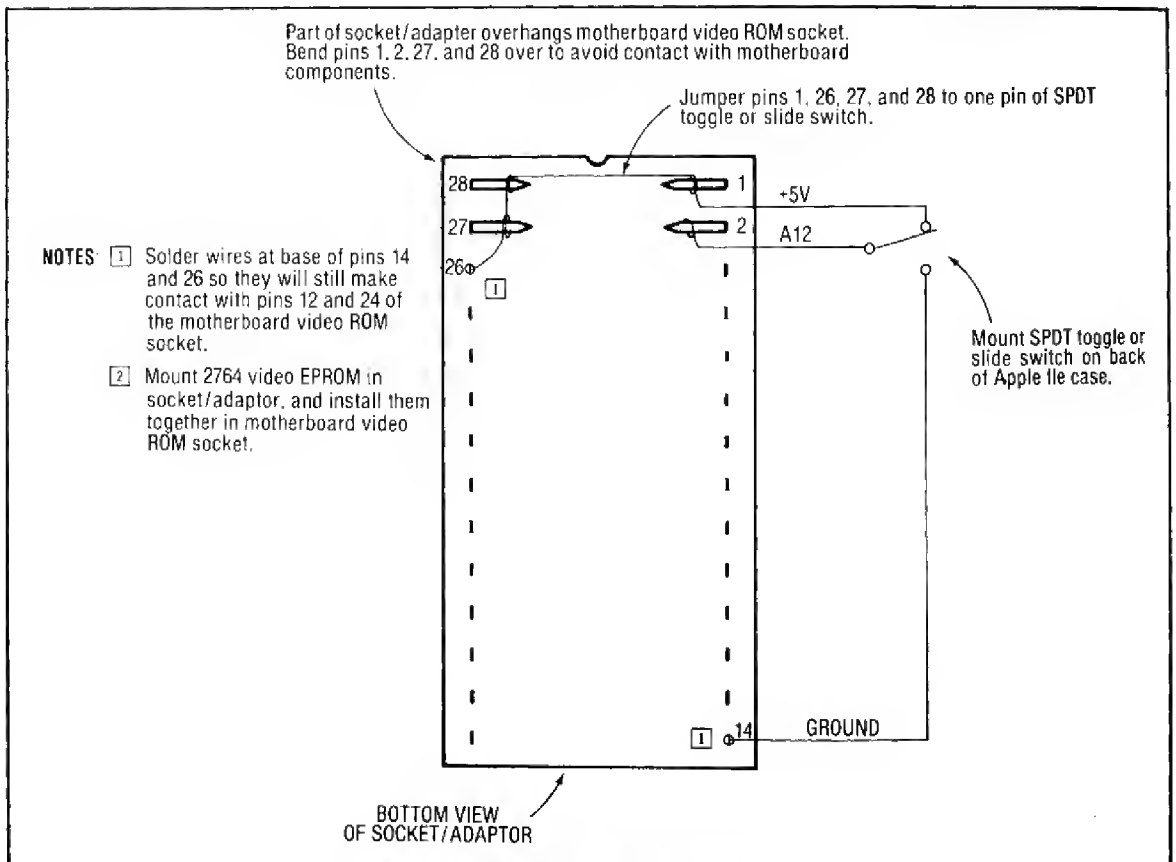


Figure 8.18 A 64K Video EPROM Adaptor.

when the resulting video EPROM is installed in the Apple IIe.

What's wrong with the displayed text characters the way they are? Nothing—it's just that customizing your Apple is fun and rewarding. Burning your special video EPROM is an easy way to do this, and the capability is a nice feature of the Apple IIe.

A final word of advice is to use 350-nanosecond EPROM for your video EPROM rather than the more commonly available 450-nanosecond variety. The Apple IIe design only gives a 390-nanosecond address setup time before the ROM data must be valid. 450-nanosecond EPROM will probably work, but if your screen display starts looking odd on a hot summer day, you'll know what's causing it.

```
 10  REM
 30  REM  PREPARE GRAPHICS AREA OF IIE VIDEO PROM
 40  REM
 50  REM          BY JIM SATHER --- 6/24/83
 70  REM
100  REM  CHANGE HØ'TABLE TABLE TO REASSIGN THE
110  REM  LORES COLORS TO DIFFERENT NUMBERS.
120  REM
130  REM  HØ'TABLE
140  REM
150  DATA  Ø,17,34,51,68,85,1Ø2,119,136,153,17Ø,187,2Ø4,221,238,255
160  DIM LOTABLE(15): FOR A = Ø TO 15: READ LOTABLE(A): NEXT
200  REM
210  REM  HØTABLE:
230  REM  THIS TABLE CONTAINS THE LORES PATTERNS (ROTATED TWO BITS).
240  REM
250  DATA  Ø,68,136,204,17,85,153,221,34,1Ø2,17Ø,238,51,119,187,255
260  DIM HØTABLE(15): FOR A = Ø TO 15: READ HØTABLE(A): NEXT
400  REM
410  REM  THESE LOOPS GENERATE THE GRAPHICS AREA OF THE VIDEO ROM.
420  REM
430 HIRES = 256:VC = 16: TEXT : HOME
440  FOR A = 1Ø24Ø TO 12160 STEP 128: REM BUILD AT $2800
450 VC = VC - 1:NOVC = 16
460  FOR B = A TO A + 12Ø STEP 8
470  VTAB 12: PRINT B - 1Ø24Ø
480 HIRES = HIRES - 1: IF HIRES < 128 THEN HIRES = HIRES + 128
490  POKE B,HIRES: POKE B + 1,HIRES: POKE B + 4,HIRES: POKE B + 5,HIRES
500 NOVC = NOVC - 1:WRK = LOTABLE(NOVC)
510  POKE B + 2,WRK: POKE B + 3,HØTABLE(WRK / 17)
520 WRK = LOTABLE(VC): POKE B + 6,WRK: POKE B + 7,HØTABLE(WRK / 17)
530  NEXT B: NEXT A
600  REM
610  REM THE GRAPHICS AREA IS PREPARED --- BSAVE "VIDROM/2,A$2800,L$800"
620  REM
630  PRINT  CHR$ (4);"UNLOCK VIDROM/2"
640  PRINT  CHR$ (4);"DELETE VIDROM/2"
650  PRINT  CHR$ (4);"BSAVE VIDROM/2,A$2800,L$800"
660  PRINT  CHR$ (4);"LOCK VIDROM/2"
670  PRINT : PRINT
680  PRINT "THE TOP HALF OF THE VIDEO ROM NOW"
690  PRINT "RESIDES IN DISK FILE, VIDROM/2."
700  END
```

Figure 8.19 BASIC Listing: Prepare GRAPHICS Area of Apple IIe Video PROM.

SOFTWARE APPLICATION

PROGRAMMING DOUBLE-RES GRAPHICS DISPLAYS IN BASIC

The DOUBLE-RES GRAPHICS modes offer improved resolution over SINGLE-RES GRAPHICS at a cost of greater memory usage and slower display update. This superior capability doesn't exist in the Apple II, and it appears to have been an afterthought in the Apple IIe, missing as it was from from Revision A and lacking as it is in firmware support. The Applesoft HPLOT command and Applesoft and Integer PLOT, HLINE, and VLINE commands support only the GRAPHICS modes that are present in both the Apple II and Apple IIe, not the DOUBLE-RES capabilities of the Apple IIe.

Development and presentation of comprehensive machine language plotting utilities would be a sizeable task, beyond the scope of *Understanding the Apple IIe*. However, it is possible to write some very simple Applesoft subroutines to plot 560 x 192 HIRES points and 80 x 48 LORES blocks. Plotting this way is slow, as you have probably guessed, but it is a fairly painless way of producing DOUBLE-RES GRAPHICS displays from Applesoft.

The basic technique is for a program to compute the 560 x 192 or 80 x 48 H,V coordinates, then call the plotting subroutine which develops the correct horizontal coordinate and plots the point or block in auxiliary card or motherboard memory using HPLOT or PLOT. It is easy for the subroutine to choose motherboard or auxiliary card memory for plotting via PAGE2 with 80STORE set.

Figure 8.20 is an example program which produced the LORES80 concentric circles of Figure 8.12. Initialization begins at line 100 and consists of setting up LORES80 NOMIX mode and clearing the screen. Routines beginning at line 200 compute colors and 80 x 48 H,V coordinates of concentric circles and call the plot subroutine. The computations are sin/cos manipulations with the H-coordinate scaled by a factor of 1.36 to compensate for Apple aspect ratio. The plot subroutine itself begins at line 1000.

80STORE is set at the beginning of this program, and it remains set throughout the program. Auxiliary memory plotting can then be selected via PAGE2 (POKE AUX,0), and motherboard memory plotting can be selected by PAGE2′ (POKE MBD,0). The screen is cleared by calling the monitor CLRSCR routine ($F832) after POKE AUX,0, then again after POKE MBD,0.

Odd horizontal blocks are plotted in motherboard memory at H/2 in whatever color is selected by the LORES COLOR= command. Even horizontal blocks are plotted in auxiliary card memory at H/2 in a transformed color equivalent to the color selected by COLOR. The transformation consists of getting the current color from $30 and rotating it right one bit. The auxiliary block color transformation occurs in lines 1050 and 1060. After the plotting is accomplished, $30 is restored to the color selected by COLOR=.

The LORES80 plotting is based on the fact that on a scale of 0 to 79, even blocks are plotted in auxiliary card memory and odd blocks are plotted in motherboard memory. Hires plotting is a little more difficult because on a scale of 0 to 559, points at 560 MOD 14 = 0 to 6 are plotted in auxiliary card memory, and points at 560 MOD 14 = 7 to 13 are plotted in motherboard memory (curse Apple for not including a MOD statement in Applesoft). Still, the 560 x 192 HIRES plot subroutine (see lines 1000—1060 of Figure 8.21) is no big deal.

Figure 8.21 is the program that produced the HIRES colored sine waves of Figure 8.16. It is more involved than the LORES example, but not because 560 x 192 plotting is more complex. To the contrary, 560 x 192 plotting is simpler than LORES 80 x 48 plotting because it is monochrome. The reason the HIRES program is more involved than the LORES program is that it demonstrates two different methods of constructing 4-dot HIRES colors from the 560-point monochrome subroutine.

The HIRES program plots sine waves at color, C, equivalent to LORES colors 0—15. The subroutine at line 2000 converts C to its binary equivalent in C(3)—C(0). HIRES80 to LORES color equivalency is such that 560 MOD 4 = 0, 560 MOD 4 = 1, 560 MOD 4 = 2, and 560 MOD 4 = 3 points are equivalent to C(1), C(2), C(3), and C(0) respectively of the binary color word. For example, if HIRES80 points 0, 2, and 3 are turned on the resulting color will be equivalent to LORES color C(1), C(2)′, C(3), C(0) = 1011 = $B.

In the first color construction method (see left side of Figure 8.16), there are 140 plotted horizontal positions. The first position consists of dots 0, 1, 2, and 3; the second position consists of dots 4, 5, 6, and 7; and so forth. At each plotted position, the combination of dots determined by C(1), C(2), C(3), and C(0) is turned on to produce a color equivalent to LORES color C. For a given coordinate H,V, the program steps that plot these 4-dot positions are at

```
10  REM
20  REM
30  REM                    LORES80 CONCENTRIC CIRCLES
40  REM
50  REM                    BY JIM SATHER --- 6/21/84
60  REM
70  REM
100  REM  INITIALIZE
110  REM
120 STR80 =  - 16383:COL80 =  - 16371:NOMIX =  - 16302:MBD =  - 16300
130 AUX =  - 16299:FRCTXT =  - 16290:LCLEER =  - 1998:CLR = 48
140  GR : POKE STR80,0: POKE NOMIX,0: POKE COL80,0: POKE FRCTXT,0
150  POKE AUX,0: CALL LCLEER: POKE MBD,0: CALL LCLEER: REM CLEAR SCREEN
200  REM
210  REM           COLOR / CIRCLE COMPUTATIONS
220  REM
230 PTS = 208: REM NUMBER OF ANGLES SAMPLED
240  FOR ANGLE = 0 TO PTS - 1
250 A1 = 6.2831852 * ANGLE / PTS:SI =  SIN (A1):CO =  COS (A1)
260  FOR C = 0 TO 7: COLOR= C: IF ANGLE > PTS / 2 THEN  COLOR= C + 8
270  IF C = 0 THEN  COLOR= 15
280 R = 24 - C * 3:V = 24 - SI * R:H = 40 + CO * R * 1.36: GOSUB 1010
290  NEXT C: NEXT ANGLE
300  GOTO 300
1000  REM
1010  REM           PLOT H,V      (H = 0-79, V = 0-47)
1020  REM
1030 H2 =  INT (H) / 2
1040  IF H2 -  INT (H2) THEN  POKE MBD,0: PLOT H2,V: GOTO 1080
1050  POKE AUX,0:CSAV =  PEEK (CLR):C2 = CSAV / 2:C2% = C2
1060  IF C2 - C2% THEN C2% = C2% + 128
1070  POKE CLR,C2%: PLOT H2,V: POKE CLR,CSAV
1080  RETURN
```

Figure 8.20 BASIC Listing: LORES80 Concentric Circles.

lines 370—400.

The second color construction method (see right side of Figure 8.16) supports 557 dot positions, each four dot-widths wide. For color coherency, it must be determined where the position starts in relation to COLOR REFERENCE. This is easy since we begin with the information that position 0 starts with a C(1) dot. For example, 379 MOD 4 = 3, so position 379 starts with a C(3) dot and consists of C(3), C(0), C(1), and C(2) dots in order. Plotting color $B at position 379 therefore consists of turning on dots 379, 380, and 381 ($B = 1011 = C(3), C(2)', C(1), C(0)). For a given coordinate H,V, the program steps that plot these 4-dot positions are at lines 550—590.

Lines 100—180 of Figure 8.21 initialize the Apple IIe HIRES80 display mode. Note that a short machine language routine is poked into memory that loads $20 to the accumulator and jumps to $F3EA (lines 150—170). Calling this short routine will result in clearing of the $2000—$3FFF area without setting or resetting PAGE2. This provides a fast way of clearing the HIRES80 display (POKE AUX,0 : CALL HCLEER : POKE MBD,0 : CALL HCLEER).

Of the two color construction methods, the 140 x 192 method is generally faster, but the 557 x 192 method yields higher resolution pictures with colors having more than one dot in the pattern. Either method can be utilized to produce HIRES pictures in any LORES color. These routines, however, are useful only for experimentation, education, and plotting where Applesoft speed is sufficient. Faster DOUBLE-RES plotting requires machine language computation and plotting.*

*See "True Sixteen-Color Hi-Res" by Allen Watson III, *Apple Orchard*, January 1984, for an assembly language listing of some HIRES80 plotting routines. The source code and some demonstration programs are available on diskette by sending $15 to Apple Orchard, PO Box 6502, Cupertino, CA 95015.

```
10  REM
20  REM
30  REM                    HIRES80 SINEWAVES
40  REM
50  REM               BY JIM SATHER --- 6/21/84
60  REM
70  REM
100  REM INITIALIZE
110  REM
120 STR80 =  - 16383:COL80 =  - 16371:NOMIX =  - 16302:MBD =  - 16300
130 AUX =  - 16299:FRCTXT =  - 16290:HCLEER = 768
140  HGR : POKE STR80,0: POKE NOMIX,0: POKE COL80,0
145  POKE FRCTXT,0: HCOLOR= 3
150  REM MOVE "LDA #$20, JMP $F3EA" TO HCLEER
160  POKE HCLEER,169: POKE HCLEER + 1,32: POKE HCLEER + 2,76
170  POKE HCLEER + 3,234: POKE HCLEER + 4,243
180  POKE AUX,0: CALL HCLEER: REM CLEAR AUX
190  HPLOT 140,0 TO 140,26: HPLOT 140,167 TO 140,191: REM DRAW VERT LINE
200  POKE MBD,0: HPLOT 139,0 TO 139,26: HPLOT 139,167 TO 139,191
300  REM
310  REM          GENERATE 140 X 192 SINEWAVES
320  REM
330  FOR C = 1 TO 15:CLR = C: GOSUB 2000: REM FOR COLOR=1TO15;GET BINARY
340  FOR H0 = 0 TO 276 STEP 4: REM LEFT SCREEN; 1 OF 4 POINTS
350  FOR HPART = 0 TO 2: REM PLOT V FOR H, H+1.3, H+2.7 TO SMOOTH OUT
360 V = 24.5 + C * 9 - 32 *  SIN ((H0 + 1.333333 * HPART) * .0224399)
370  IF C(1) THEN H = H0: GOSUB 1010: REM BLUE DOT
380  IF C(2) THEN H = H0 + 1: GOSUB 1010: REM BLUE-GREEN DOT
390  IF C(3) THEN H = H0 + 2: GOSUB 1010: REM BROWN DOT
400  IF C(0) THEN H = H0 + 3: GOSUB 1010: REM MAGENTA DOT
410  NEXT HPART: NEXT H0
500  REM
510  REM          GENERATE 560 X 192 SINEWAVES
520  REM
530  REM  SIN (2*PI*H/280); RIGHT SIDE
540  FOR H0 = 280 TO 556:V = 24.5 + C * 9 - 32 *  SIN (H0 * .0224399)
550 PRT4% = (H0 / 4 -  INT (H0 / 4)) * 4 + .5: REM PRT4% = H0 MOD 4
560  IF PRT4% = 0 AND C(1) OR PRT4% = 1 AND C(2) OR PRT4% = 2 AND C(3) OR
     PRT4% = 3 AND C(0) THEN H = H0: GOSUB 1010: REM PLOT POSITION+0
570  IF PRT4% = 0 AND C(2) OR PRT4% = 1 AND C(3) OR PRT4% = 2 AND C(0) OR
     PRT4% = 3 AND C(1) THEN H = H0 + 1: GOSUB 1010: REM PLOT POSITION+1
580  IF PRT4% = 0 AND C(3) OR PRT4% = 1 AND C(0) OR PRT4% = 2 AND C(1) OR
     PRT4% = 3 AND C(2) THEN H = H0 + 2: GOSUB 1010: REM PLOT POSITION+2
590  IF PRT4% = 0 AND C(0) OR PRT4% = 1 AND C(1) OR PRT4% = 2 AND C(2) OR
     PRT4% = 3 AND C(3) THEN H = H0 + 3: GOSUB 1010: REM PLOT POSITION+3
600  NEXT H0: NEXT C: END
1000  REM
1010  REM          PLOT H,V      (H = 0-559, V = 0-191)
1020  REM
1030 MOD14 =  INT (H) / 14:MOD14% = MOD14
1035 PART14% = 14 * (MOD14 - MOD14%) + .5
1040  POKE AUX,0: IF PART14% > 6 THEN  POKE MBD,0:PART14% = PART14% - 7
1050  HPLOT MOD14% * 7 + PART14%,V
1060  RETURN
2000  REM
2010  REM  CONVERT CLR FROM DECIMAL TO HEX
2020  REM
2030 CLR% = CLR: FOR A = 0 TO 3
2040 C(A) = 1:CLR = CLR% / 2:CLR% = CLR% / 2
2045  IF CLR = CLR% THEN C(A) = 0
2050  NEXT : RETURN
```

Figure 8.21 BASIC Listing: HIRES80 Sine Waves.

TECHNICAL NOTE

DETAILS OF TELEVISION PROCESSING OF APPLE VIDEO

A rigorous examination of the television processing of the Apple signal involves technical details beyond the scope of *Understanding the Apple IIe*. Brief descriptions of some of these technical details are presented here for those readers who wish to study television processing of Apple IIe video in depth.

A square wave is the sum of the odd harmonics of a sine wave of the same frequency. For example, a 3 MHz square wave can be produced by summing the following sine waves:

3 MHz at amplitude A
9 MHz at amplitude A/3
15 MHz at amplitude A/5
.
.
.

The more harmonics added, the more perfect the square wave. This sinusoidal make-up of a square wave is significant because tuned circuits such as those found in a television receiver respond to the sinusoidal components of signals. A square wave will be processed as the sum of its sinusoidal components.

Generally, Apple PICTURE signals, which produce color displays, are 3.58 MHz square waves. These square waves modulate a television carrier frequency in the user supplied modulator, creating a radio frequency with a square modulation envelope. Sinusoidally, the square wave intelligence is carried by the following series of frequencies:

carrier
carrier + 3.58 MHz
carrier + (3.58 MHz) x 3
carrier + (3.58 MHz) x 5
.
.
.

The IF strip of the television will pass the sine wave carrier and those sine wave frequencies above the carrier, up to carrier + 4.2 MHz. Only the carrier and carrier + 3.58 MHz of the above distribution are within this range. As a result, the 3.58 MHz square envelope is converted to a 3.58 MHz sinusoidal envelope, and the output of the second detector in the television is a 3.58 MHz sine wave. This sine wave is passed by the chrominance amplifier to the synchronous demodulator, where it is processed with the reconstructed COLOR REFERENCE sine and cosine waves to produce color signals. It is also processed by the luminance amplifier to produce the luminance signal.

Many televisions have a 3.58 MHz trap in the luminance path which reduces color interference with the luminance signal. The effect of this trap is to remove the 3.58 MHz variation, and pass a gray luminance level which lasts for the duration of the 3.58 MHz presence. A similar effect is felt on the 7 MHz modulation envelope produced by LORES/HIRES80 5 and 10 colors. The 7 MHz + carrier frequency is out of the band pass of the IF strip, so the 7 MHz variations are removed and replaced by a gray level. These solid gray levels do not degrade the Apple luminance signal, but enhance it. We cannot see 3.58 MHz variations in picture brightness at normal viewing distance. We just see solid blocks of brightness. Conversion of 7 MHz and 3.58 MHz signals to solid gray levels does not, therefore, degrade the picture we perceive.

A very interesting special case among Apple PICTURE signals is that created by turning alternating groups of three HIRES40 dots on and off. Conventional Apple wisdom is that this will create a horizontal dashed line with white coloration of the dashes because they are three adjacent dots. However, the picture signal produced by this pattern is a square wave of 3.58 MHz/3. This square wave has significant 3.58 MHz sinusoidal content, since 3.58 MHz is the third harmonic of the fundamental square wave frequency. This produces a 3.58 MHz sine wave at the output of the chrominance amplifier about 1/3 the amplitude of the signal produced by a 3.58 MHz PICTURE signal. The result is a washed out coloring of the 3.58 MHz/3 PICTURE signal, not nearly as intense as the coloring of 3.58 MHz PICTURE signals. The chrominance amplifier frequency band is from 3.1 MHz to 4.1 MHz, so any PICTURE signal from 3.1 MHz/3 to 4.1 MHz/3 should have some coloration.

A second television phenomenon is less predictable. Many televisions have a coupling transformer or inductor/capacitor combination at the input to the chrominance amplifier. I have found that this input circuit has a marked tendency to ring at 3.58

MHz when the PICTURE signal switches from white to black or black to white. This ringing produces an output from the chrominance amplifier of about the same amplitude as that produced by a 3.58 MHz/3 PICTURE signal. One result is edge coloring of white screen displays. The 3.58 MHz ringing should vary greatly from television to television, and may be reponsible for many of the off-color edges found in Apple displays.

In a normal television broadcast signal, the luminance signal energy is concentrated at multiples of the horizontal frequency removed from the picture carrier. This is because the luminance signal itself has a very high content of harmonics of the line scanning frequency. When the luminance signal modulates the carrier, the energy is largely distributed in groups centered at the following frequencies:

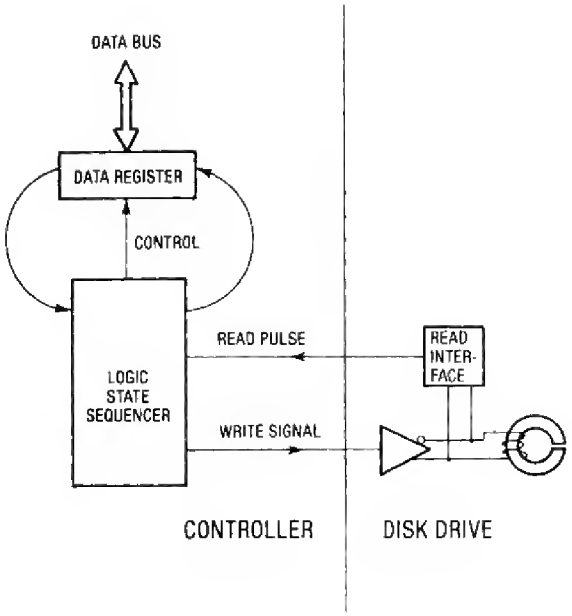
carrier
carrier + horizontal frequency
carrier + 2(horizontal frequency)
.
.
.

The color signals have a similarly high content of horizontal frequency harmonics. When the 3.58 MHz color subcarrier is modulated by a color signal, the energy is largely distributed at

3.58 MHz (suppressed)
3.58 MHz + horizontal frequency
3.58 MHz + 2(horizontal frequency)
.
.
.

The 3.58 MHz color subcarrier is used because carrier + 3.58 MHz resides midway between the "carrier + 227(horizontal freq)" and the "carrier + 228(horizontal freq)" energy concentrations of the luminance signal. The "carrier + n(horizontal freq)" luminance distribution is thus interlaced with the "carrier + 227.5(horizontal freq) + n(horizontal freq)" distribution. This reduces interference between the chrominance and luminance signals.

The Apple PICTURE signal is high in horizontal frequency harmonic content, just like the normal television luminance signal and color signals. Therefore, the energy of the modulated carrier should be distributed primarily at "carrier + n(horizontal freq)" intervals. The process of modulating a 3.58 MHz subcarrier with a color signal does not take place, so the "carrier + 3.58 MHz + n(horizontal freq)" energy distribution should not exist to the same extent as it does in a normal television chrominance signal. However, the 3.58 MHz PICTURE signal with horizontal frequency harmonic periodicity should create some elements of the "carrier + 3.58 MHz + n(horizontal freq)" distribution when the carrier is modulated. It is, therefore, interesting to note that the 3.58 MHz COLOR REFERENCE signal of the Apple is 228 times the horizontal frequency, not 227.5 times the horizontal frequency as in a normal television signal. This means that the "carrier + n(horizontal freq)" energy distribution is superimposed, rather than interlaced, with the "carrier + 3.58 MHz + n(horizontal freq)" distribution. My conclusion is that the energy of a carrier wave modulated by Apple video is largely distributed at multiples of the horizontal frequency removed from the carrier frequency, with no interlaced distribution induced by the chrominance element.



chapter 9

The Disk Controller

The long term success of the Apple II line of computers could not have come about without the development of the Disk II 5 ¼ inch floppy disk drive. For simple storage of data and programs, disk I/O is merely faster and more convenient than cassette I/O. But for important computer uses such as word processing, data base management, business accounting, and file handling, the disk drive or its equivalent is mandatory. There can be no doubt that for most owners the disk drive is the most important peripheral in the Apple IIe computer.

Since the original Apple II was strictly cassette based, the interface to the Disk II had to be built on a peripheral card. The extent of motherboard support of the Disk II is an empty card slot and the motherboard firmware. Motherboard firmware includes no disk handling routines but only looks for disk handling routines in the peripheral slots and jumps to them at power up. The **bootstrap** program and the circuits that interface the computer with the drive are on the **disk controller**, which is a peripheral card usually installed in Slot 6. The disk controller is connected by a 20-wire ribbon cable to the disk **drive**, which contains more electronic circuits as well as the drive mechanisms.

It is primarily the controller circuits and the available disk operating systems which determine the features of disk operation that are unique to the Apple, and it is the controller circuits which are the main topic of this chapter. General features of the disk drive are also discussed, but no attempt is made here to document the disk operating systems beyond the **RWTS** (Read or Write a Track and Sector) subroutine of DOS 3.3, the **DIIDD** (Disk II Device Driver) subroutine of ProDOS, and the disk data formats of DOS 3.2, DOS 3.3, and ProDOS.*

DISK II OVERVIEW

Near the end of 1977, Apple Computer's decision-making group was still very small. Mike Markkula, the chairman of the board, presented the group with a list of products needed to be developed for the Apple. At the top of the list was a floppy disk drive for the Apple II. Within a very short period of time,

*Operation of DOS 3.3 and ProDOS are well documented in the books *Beneath Apple DOS* (Quality Software, 1982) and *Beneath Apple ProDOS* (Quality Software, 1984) by Don Worth and Pieter Lechner.

Apple developed the Disk II and released it along with its operating system, DOS 3.

The disk drive chosen by Apple was a Shugart SA400. Apple designed their controller card (mounted in a peripheral slot), analog card (mounted in the disk drive), and data formats around this standard drive. More recently, companies other than Shugart have made the Disk II drive for Apple, but the Disk II has always been nearly identical to the old Shugart drive. Additionally, other Apple drives such as the twin half-height drive and the Apple IIc built-in drive perform identically to the Disk II. That is, they are single sided, single density drives that support 35 tracks and 143,360 bytes of data on 5 ¼ inch floppy disks.

Various companies make other drives and controllers which will work in the Apple IIe. These drives are generally less expensive than their Apple counterparts and they work perfectly well with DOS 3.3, ProDOS, and the Disk II controller. The main disadvantage of buying alternate source disk drives is that you do not receive the excellent documentation that comes with an Apple drive. This documentation, however, can be purchased separately from retailers who carry Apple products. Other compatible drives and controllers will be similar, but the discussions in this chapter detail the operation of the Disk II controller and, to a lesser extent, the Disk II drive.

Floppy disks are **magnetic media**, just like audio and video tapes. Reading and writing is performed by rotating the disk while a stationary read/write head presses against it. Disk speed is 300 RPM which translates to 48.4 inches per second on the inner track. This is a great deal faster than audio cassette tape speed, so the rate of data transfer is greater than that achieved in an audio cassette storage system.

In between read or write periods, the head may be positioned radially so that different **tracks** can be written to or read from. The head is positioned precisely by a **stepper motor** under 6502 program control. There are 35 tracks on the Disk II, but some second source drives have 40 or more tracks which can be utilized by modifying DOS 3.3 or ProDOS. There is no hardware sensor that can be used to determine which track the head is on. The bootstrap program absolutely determines where the head is by running it against the outer stop at initialization. From that point the controlling DOS closely monitors head location, always saving the current position in RAM. Also, when reading data from or writing data to a formatted disk, the controlling DOS always verifies head position by reading the stored track number from the disk and comparing it to the track number it is attempting to access.

Figure 9.1 is a functional diagram of Apple IIe disk I/O. Data transfer between the MPU and the controller is 8-bit parallel via the data bus. Control by the MPU is via the address bus, of course.* Data transfer between the controller and the drives is serial, and control of the drives is via multiple control lines serving various functions. The controller is primarily a digital data processing device, while the analog circuit card in the disk drive performs the functions of amplification, shaping, and gating. Control of the disk is software intensive, meaning very little is done automatically by hardware.

Hard sectored disks are disks with little holes in them which divide the disk into a number of sectors. Disk drives supporting hard sector formats have a sensor in them which signals the host computer when a hole is passing by and allows a program to determine where the disk is in its circular trip. The Disk II does not have this feature, and Apple disk operating systems don't require it. Instead, the Apple uses a **soft sectored** format in which positional information is stored on the disk in uniquely identifiable **address fields**. These address fields are the "holes" which divide the disk into sectors and identify rotational position. The address fields contain an address field identifier and a volume-track-sector address from which programs can locate specific address fields. Behind each address field, there is a **data field** with space for 256 bytes of data. An address field and the data field that follows it make up a **sector** of disk information.**

DOS 3.3 and ProDOS write 16 sectors (address fields followed by null data fields) in their disk formatting routines. The **16-sector format** is not unconditionally dictated by hardware. It is just a very reasonable number of sectors to have, considering the facts that the 6502 addressing modes are best suited for manipulation of data blocks up to 256 bytes in size, that 256 bytes is a workable size for data blocks in disk I/O, and that the Disk II is capable of storing 16 sectors of 256 bytes on a track in the DOS 3.3/ProDOS format. As an example, the hardware will let you store data in eight sectors of 512 bytes each. Sixteen sectors of 256 bytes, however, is

*In this chapter there is much discussion of the role of the MPU in disk I/O. The reader will do well to remember that while the MPU has certain capabilities of manipulating the Disk II hardware, these capabilities can only be utilized under program control. In other words, a 6502 program must supervise the role of the MPU in disk I/O.

**The Apple Disk II will work with hard sectored disks, but the holes in the disk will be ignored.

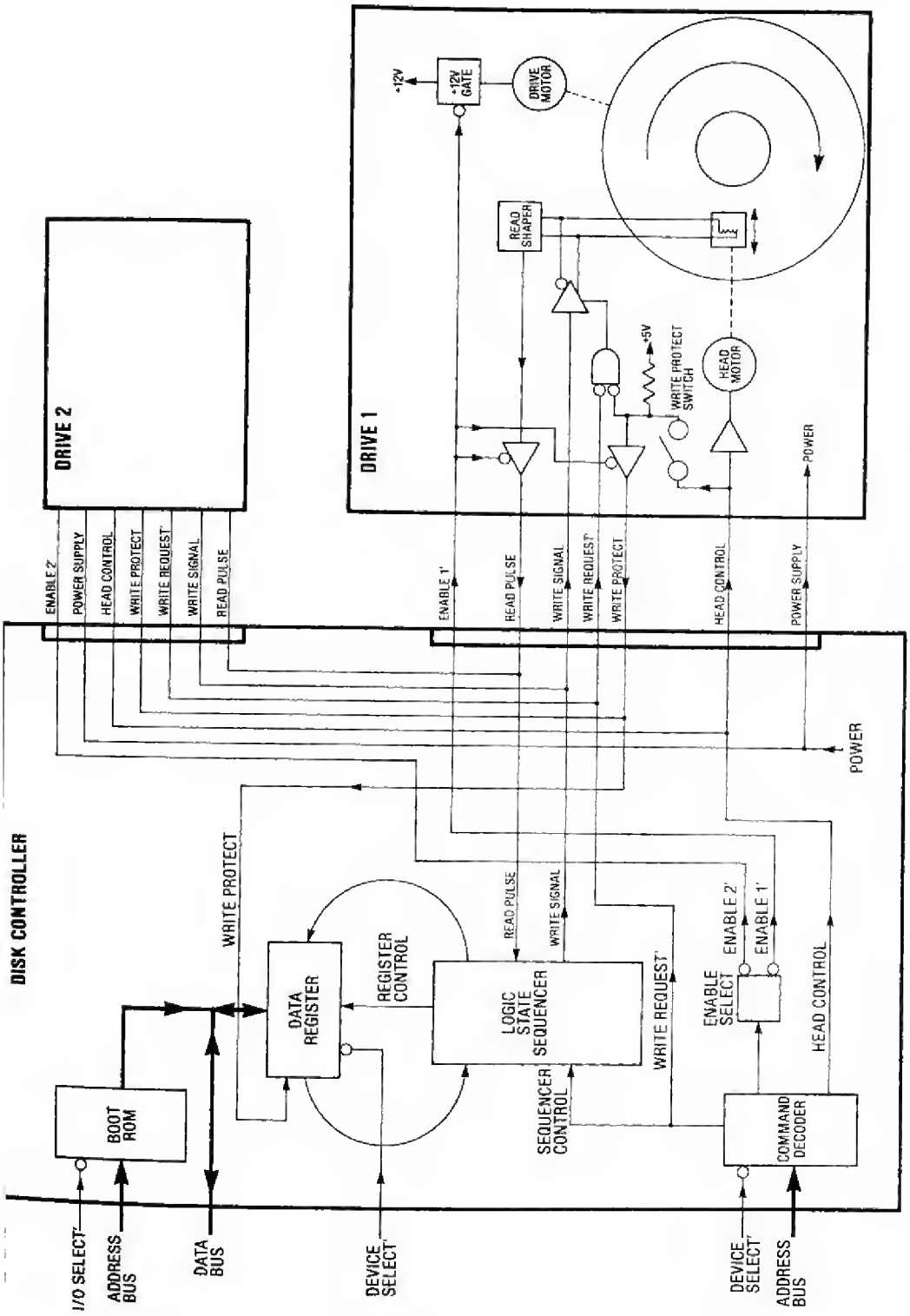


Figure 9.1 Functional Diagram: The Disk Interface.

the only format supported by the RWTS subroutine of DOS 3.3 and the Disk II device driver of ProDOS, and the only reasons to deviate from it are for copy protection or to have fun (using the word "fun" in a very broad sense).

With DOS 3.3, the nuts and bolts details of disk I/O are handled by the **RWTS** (Read or Write a Track and Sector Subroutine). The majority of DOS 3.3 is concerned with such tasks as command interpretation and execution, file management, track and sector mapping, and cataloging. Anytime it actually reads or writes disk information, however, it uses RWTS. There are four types of calls to RWTS: FORMAT, READ, WRITE, and SEEK (head position only). **FORMAT** writes identifying information for 16 sectors on all 35 tracks. **READ** positions the head and reads a specified track/sector, and **WRITE** positions the head and writes to a specified track/sector. **SEEK** moves the head to a specified track.

Functions comparable to those handled by RWTS are handled by the **DIIDD** (Disk II Device Driver) in ProDOS. Data is written or read, two sectors per call, in 512-byte data blocks. There are three types of calls to the Disk II device driver: READ block, WRITE block, and return STATUS. The **READ** and **WRITE** calls perform the same function as READ and WRITE calls to RWTS except that two sectors (512 bytes) of data are transferred instead of one. The **STATUS** call checks if a write protected disk is installed in a drive. Formatting of disks is not performed by the device driver but is performed by FILER, a separate utility associated with ProDOS.

Ignoring elaborate copy protection methods, the normal method of **reading** a sector is to position the head and poll the disk input port until the desired identifying leader, or **address field**, is found. Then the data following the address field is read into an area of RAM the size of the sectors being used. Data is written one sector at a time. The normal **writing** method is to find the pertinent sector on a formatted disk as when reading, but to overwrite the data area after the desired sector address field has been found.

Figure 9.2 shows the paths data takes during disk I/O. Data is transferred between the MPU and the **data register** on the controller, one byte at a time over the data bus.* Data is shifted serially between the controller and the disk drive under control of the

*The data register is referred to in some writings as the data latch.

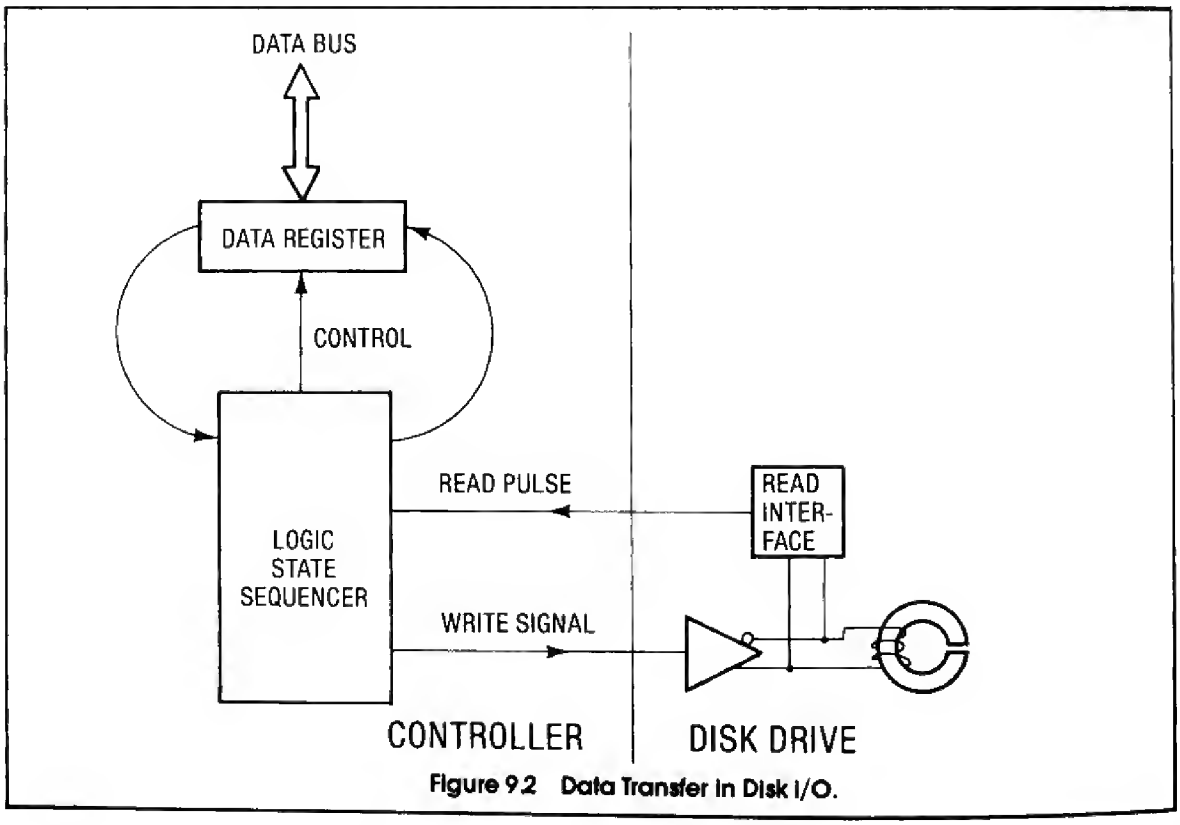


Figure 9.2 Data Transfer in Disk I/O.

logic state sequencer on the controller. The logic state sequencer is a ROM together with some flip-flops wired up to act like a little 2 MHz computer. It has a stored program which it sequences through while executing commands that control the data register. Some writings refer to the sequencer as the "state machine," but logic state sequencer more clearly describes its functions. Via address bus commands, a 6502 program can configure the sequencer to shift out write data, shift in read data, or shift in the state of the write protect switch in the disk drive.

In the **write process**, a 6502 program causes the MPU to store a byte of data in the data register of the controller. The logic state sequencer shifts this information, monitoring each bit individually. Every time the sequencer sees a ONE, it toggles the WRITE signal. This changes the direction of magnetic field in the **read/write head**. As the disk passes across the head during write operation, the surface of the disk near the head is magnetized, and the direction of field in the head determines the direction of the magnetic field on the disk. When the writing stops, the data remains on the disk in the form of transitions or lack of transitions in the magnetic field. Disk I/O is identical to cassette I/O in this regard. Serial data is stored in the form of magnetic field reversals on the medium. A ONE is a field reversal. A ZERO is the lack of a field reversal.

Reading is the reverse of writing as far as the read/write head is concerned. As the disk rotates, magnetic field reversals on the moving disk surface cause voltages to be induced in the head. The voltage pulses are sensed by a special purpose floppy disk read interface chip which puts out a nice square read pulse for every magnetic field reversal on the disk. The sequencer monitors these read pulses and shifts ONEs and ZEROs into the data register based on the presence or absence of the read pulse at the normal write interval.

The sequencer syncs up on the read data if it was written properly. This means that it will shift data into the data register until a complete byte is shifted in, then it holds that complete byte long enough for a 6502 program to detect it by polling the data register. The program recognizes that the data register holds a valid byte when the most significant bit of the register is set. When a valid byte is detected, the program must quickly process or store the byte and start checking the data register for the next one.

It can be seen from this overview that the key to understanding how data is transferred to and from the disk is the logic state sequencer and how it is manipulated by the 6502 program. Later, we will analyze the sequencer in great detail, but first we need to lay the groundwork by looking at the hardware environment.

THE DISK II DRIVE

Figure 9.3 is a functional diagram of the Disk II drive. The intention here is not to explain all details of floppy disk drive operation, but only to establish the basis of control of the drive from the computer. In addition to Figure 9.3, reference to Figure 9.1 should help clarify the points of discussion. Even though only the Apple Disk II is mentioned, most of the discussion is also valid for substitute drives. As Figure 9.1 indicates, all connections to the drive are routed to the controller.

Power Supply

The drive takes its power supply voltages from the Apple's main power supply. +12V, −12V, and −5V are all utilized, but only +12V is used for motor drive. Since +12V is used both to position the head and rotate the disk, the load on +12V is significant, especially at disk start-up. The drive has a high capacity +12V input filter to assist the Apple's power supply in supplying motor start-up current.

The Drive ENABLE' Input

The ENABLE' input is low at drive 1 or drive 2, but not both, when a drive is being accessed. At the enabled drive, the drive motor is on, the IN USE indicator glows, head positioning is enabled, and the read pulse output and the state of the write protect switch are enabled to the controller. Speed of disk rotation is regulated by a motor speed control board in the back of the drive. This speed is adjustable via a potentiometer on the speed regulator board.

The Head Positioning Mechanism

The head assembly is positioned precisely, via a stepper motor. The stepper motor, which rotates, is linked mechanically to the head assembly, which travels linearly. A 6502 program positions the head assembly by directly controlling the **four phased inputs** to the stepper motor.

Figure 9.4 is a functional diagram of a 4-phase stepping motor, which can provide a basis for understanding the positioning of the read/write head. The rotor is a cogged ferrous drum whose cogs may be attracted by one of four electromagnets. The electromagnets are activated sequentially under program control. There may be any number of cogs on the rotor, but only one of them is next to one of the four electromagnets at any time.

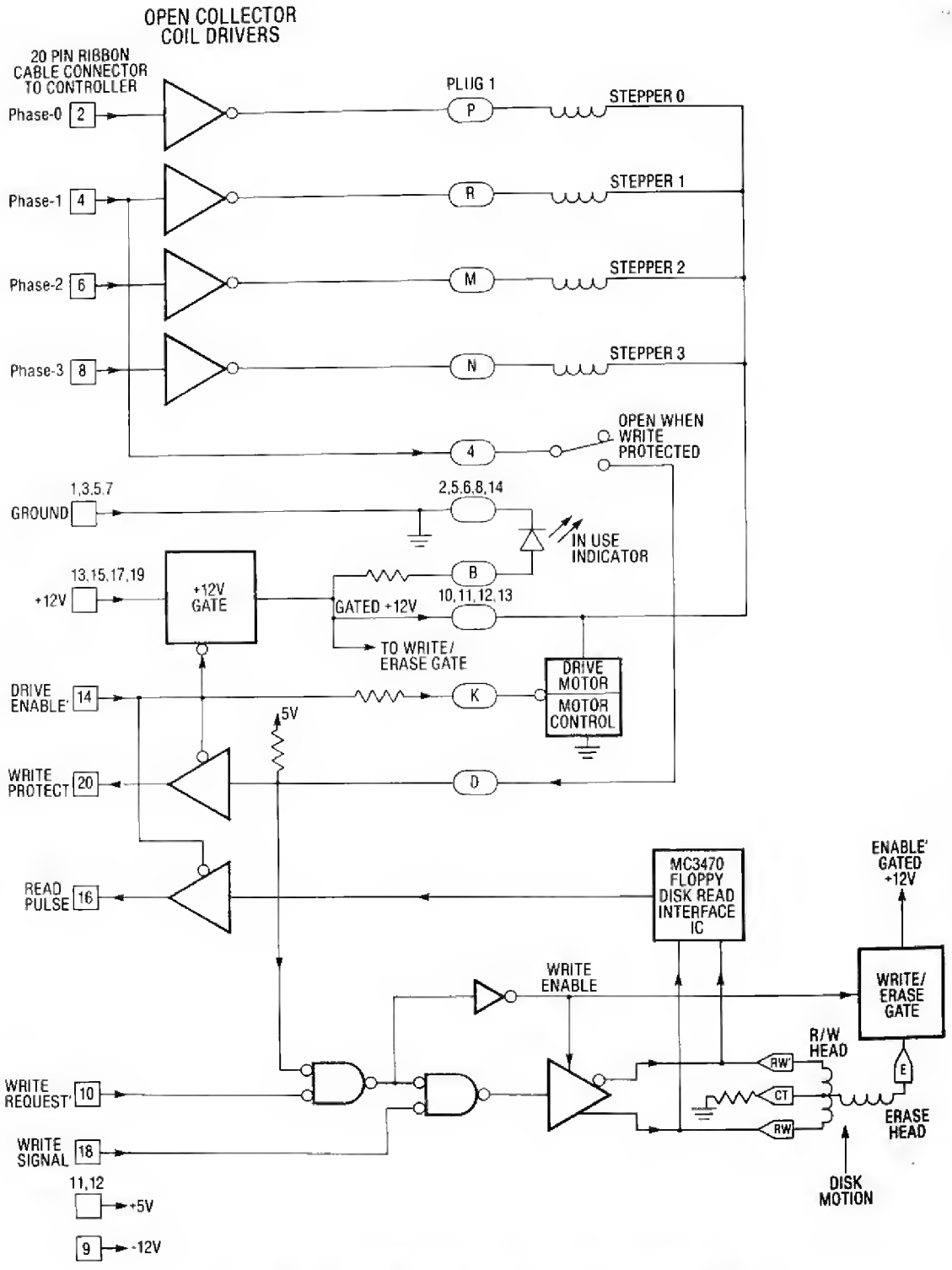


Figure 9.3 Functional Diagram: The Disk II Drive.

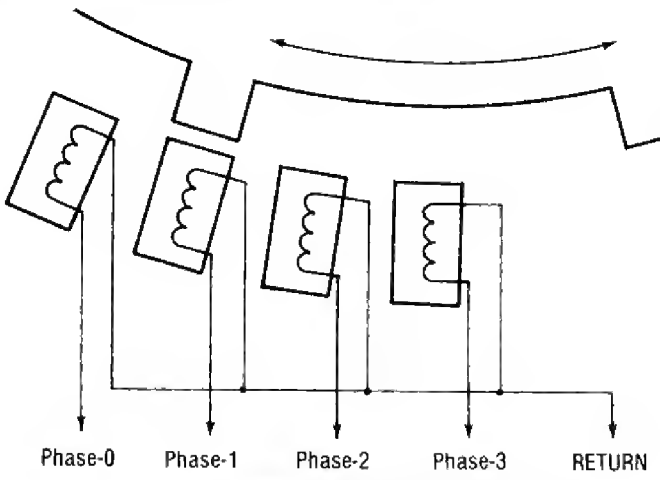


Figure 9.4 A Stepping Motor.

To rotate the rotor one step to the right in Figure 9.4, energize the phase-2 magnet, then deenergize the phase-1 magnet. Magnetic attraction will then align the cog with the phase-2 magnet. To go three steps left from this new position, perform this progression: phase-1 on, phase-2 off, wait, phase-0 on, phase-1 off, wait, phase-3 on, phase-0 off, wait. The **wait** is for the **response time** of the motor, which is slow compared to the MPU. Summarizing, to step left, sequence through the phases in descending order. To step right, sequence in ascending order.

The Apple disk drive uses a 4-phase stepper motor for head positioning. The controller has provisions for bringing the control voltage for each phase high or low individually. In the drive, these voltages will turn on or cut off current to the electromagnets in the stepper motor. A good deal of the software overhead is required to position the head and remember its location (see **PROGRAMMING EXAMPLES FROM RWTS** later in this chapter). Two phases must be stepped through to move the head one track, and the Disk II and standard versions of RWTS and DIIDD support 35 tracks on a disk.

Writing to the Disk

There are two write related inputs from the controller to the drive, WRITE REQUEST' and the WRITE signal. **WRITE REQUEST'** causes the drive to be configured for writing unless the disk is write protected. Configuring the drive for writing consists of allowing the **WRITE signal** to control the current in the coil of the read/write head, and of applying a direct current to a second head referred to here as the **erase** head. The WRITE SIGNAL control of the current in the read/write head is such that the high/low state of the WRITE signal determines the direction of the magnetic field set up in the read/write head.

Current in the read/write head tends to produce magnetic fields on the disk parallel to disk motion. Current in the erase head tends to produce magnetic fields on the disk perpendicular to disk motion. The actual strength and direction of the fields produced are a result of the vector sum of the WRITE signal field and the erase field (see Figure 9.5). The presence of the erase field means that there is always some field in the head assembly while writing, even in the middle of a WRITE signal field reversal. The absence of a field in the head assembly would allow previous field alignment on the disk to remain unchanged. Thus, the erase field causes the previously written data to be erased when the WRITE signal component of the field vector has an amplitude of zero.

As was mentioned in the overview, data bits are stored on the disk as field reversals, or the lack of field reversals, at a regular interval. This fact is not changed by the presence of the erase component in the field. Just understand that the erase, or radial, component is constant, while the read/write, or tangential, component reverses depending on the WRITE signal.

The Write Protect Switch

There is a spring loaded switch in the Disk II drive which is open when a write protected disk is installed. A write protected disk is one with no notch on the left edge or one with the notch covered up. Having this switch open causes the WRITE PROTECT signal to be high, which isolates the WRITE

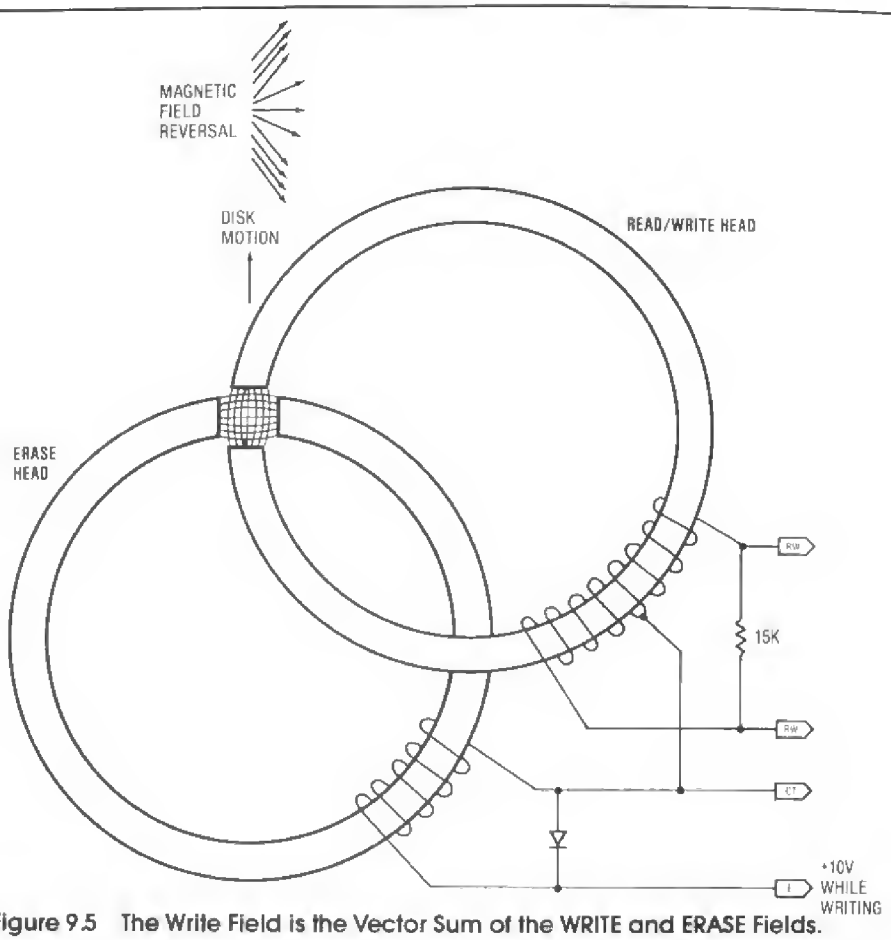


Figure 9.5 The Write Field is the Vector Sum of the WRITE and ERASE Fields.

signal from the read/write head even if WRITE REQUEST' is low. The WRITE PROTECT signal is also routed from the enabled drive to the controller so a 6502 program can determine the state of the write protect switch.

The WRITE PROTECT signal and the read pulse are the only two outputs from the drive to the controller (see Figures 9.1 and 9.3). Both are output through tri-state drivers which are gated by the drive ENABLE' input. This allows the two drives to share control of the read pulse and write protect lines.

An interesting feature of the write protect circuiry is that its activating voltage is the phase-1 voltage input to the head positioning motor (see PLUG 1, pin 4 in Figure 9.3). As a result, phase-1 must be turned off after head positioning or writing to the disk is impossible. This is probably a way of forcing the programmer to turn off all phases (not just phase-1) after head positioning. This seems to imply that keeping a stepper motor input active causes an undesirable effect—perhaps a tiny vibration or overheating of the motor. In any case, the RWTS and DIIDD head positioning routines leave all four phases off after head positioning. The boot routine on the controller card, however, leaves phase-0 energized after sending the head to track 0.

The Read Pulse

When writing is not enabled, passing a field reversal on the surface of the disk over the read/write head induces a voltage in the coil. This induced voltage will alternate in polarity for every field reversal. The induced voltage is sensed by a special purpose chip which is designed for this function. The special purpose chip, a Motorola MC3470, outputs a positive 1-microsecond pulse for every field reversal (see Figure 9.6). This **read pulse** is routed from the enabled drive to the controller.

Now we come to the biggest problem with reading a disk. The signal coming off the read/write head is

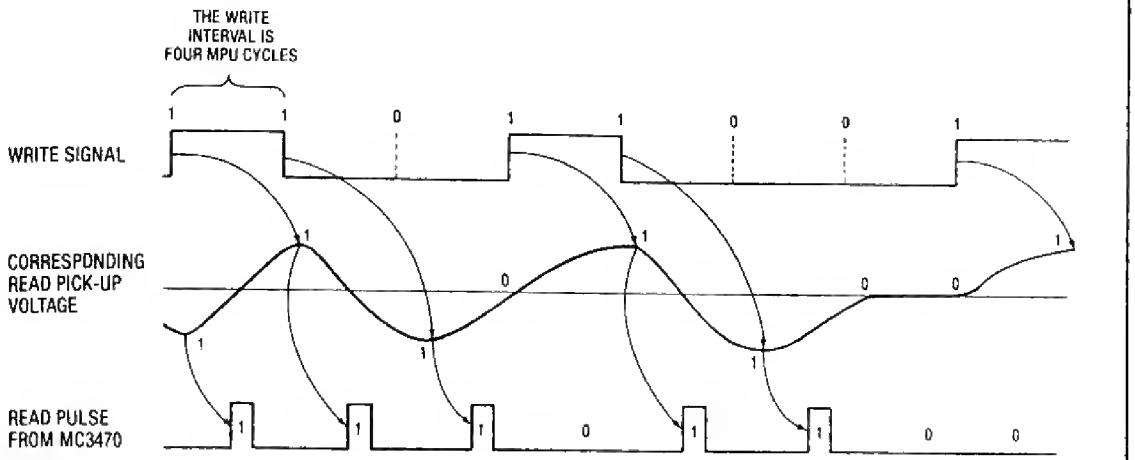


Figure 9.6 When Reading, the Lack of a Read Pulse at a Regular Interval Represents a ZERO.

a dirty little voltage. The shape and size of this **read pick-up signal** will vary with disk speed, temperature, humidity, head alignment, disk warpage, and Murphy's Law in the read environment as opposed to the same factors in the write environment. The MC3470 has to clean up the imperfect read pick-up signal and pass it to the controller. The basic features of the MC3470 clean up job include amplification, shaping, filtering, and noise rejection. The MC3470 detects positive and negative voltage peaks on the read pick-up signal, then waits about two microseconds to verify that a second opposite polarity peak has not occurred. The purpose of this is to prevent narrow noise pulses from generating invalid read pulses. After the 2-microsecond mask period, a 1-microsecond read pulse is output. Even with this sophisticated interface, the controller must be able to reliably monitor a read pulse whose timing interval will vary significantly. In addition, it must be able to interpret either the presence or the absence of the read pulse at the distorted interval.

The 2-microsecond delay period between peak detection and read pulse output turned out to be too short for the new format when Apple upgraded DOS 3.2 to DOS 3.3. Apple solved this problem by replacing fixed resistor R21 with a potentiometer on the analog card in the disk drive. Technicians align the potentiometer for an optimum delay, which works out to be approximately three microseconds.

The combination of the read/write head and the MC3470 responds very well to field reversals on the moving disk as long as there is not too much space between them. However, if there is too much space between field reversals on the disk, the MC3470 starts putting out false read pulses. This means that you can't utilize copy protect schemes that call for isolated field reversals on the disk separated by large intervals of constant field direction. In other words, the MC3470 will reliably produce read pulses while data of normal density is moving past the read/write head, but reduction of this density will cause spurious read pulses to be generated. The write interval in the Apple is four MPU cycles, and we will see that the maximum time between field reversals on a disk in any Apple DOS format is three write intervals.

THE DISK II CONTROLLER

The Disk II controller contains that part of disk I/O electronics which needs to be positioned close to the motherboard. This includes a Bootstrap ROM, a data register, a logic state sequencer, and a command decoder. Figure 9.7 is a functional block diagram of the controller, and Figure 9.8 is a full schematic.

The Bootstrap ROM

The **Bootstrap ROM** is referred to as the P5 ROM by Apple. It contains a 256-byte program which begins the bootstrap procedure that reads the DOS from a disk, puts it into RAM, and initializes it. This 256-byte program is the only 6502 program which resides on the controller, and it may be accessed any time at CnXX where n is the controller slot number. This program has just enough code to get the contents of track 0, sector 0 into RAM. Then program control is passed to that part of RAM so that the

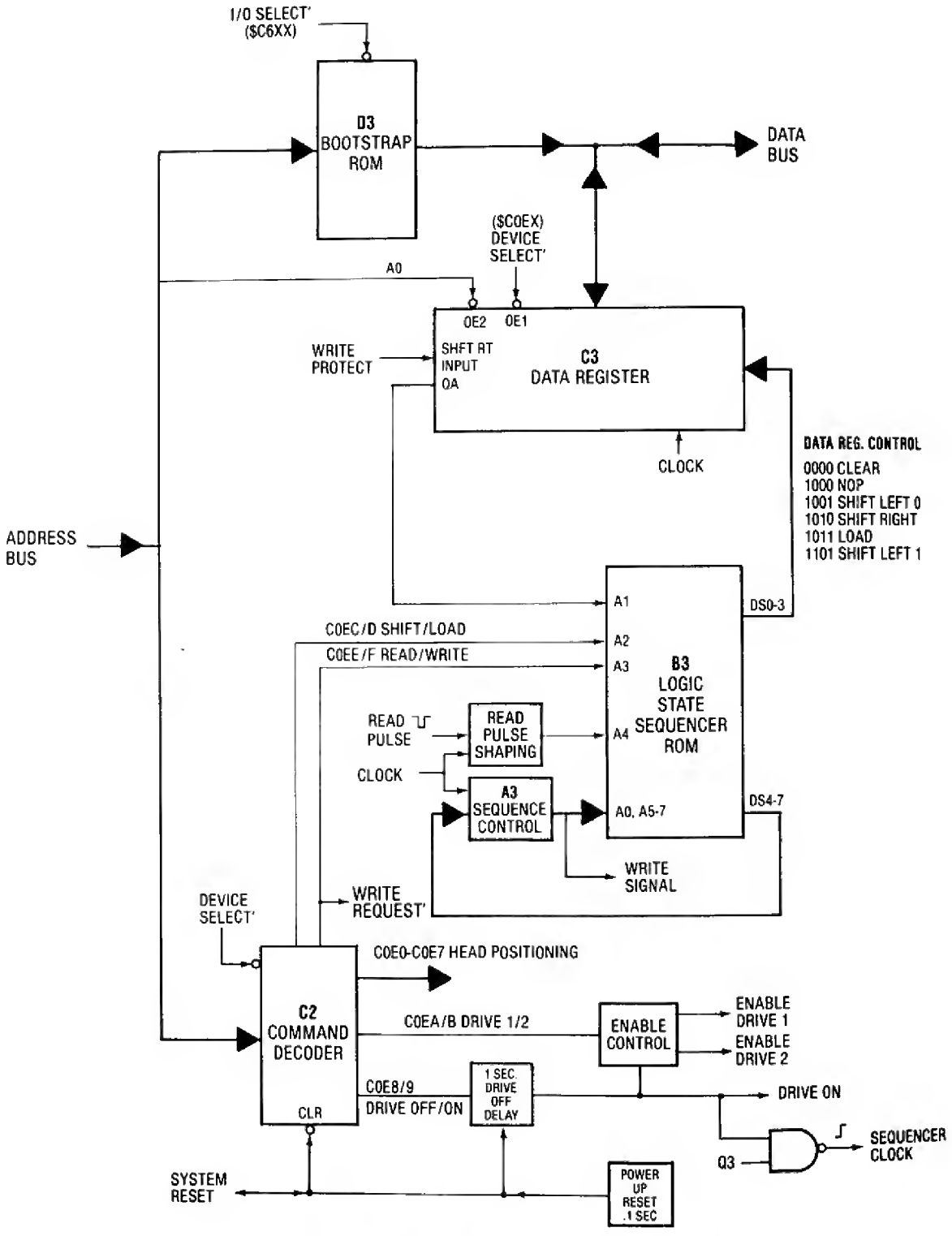


Figure 9.7 Functional Diagram: The Disk II Controller.

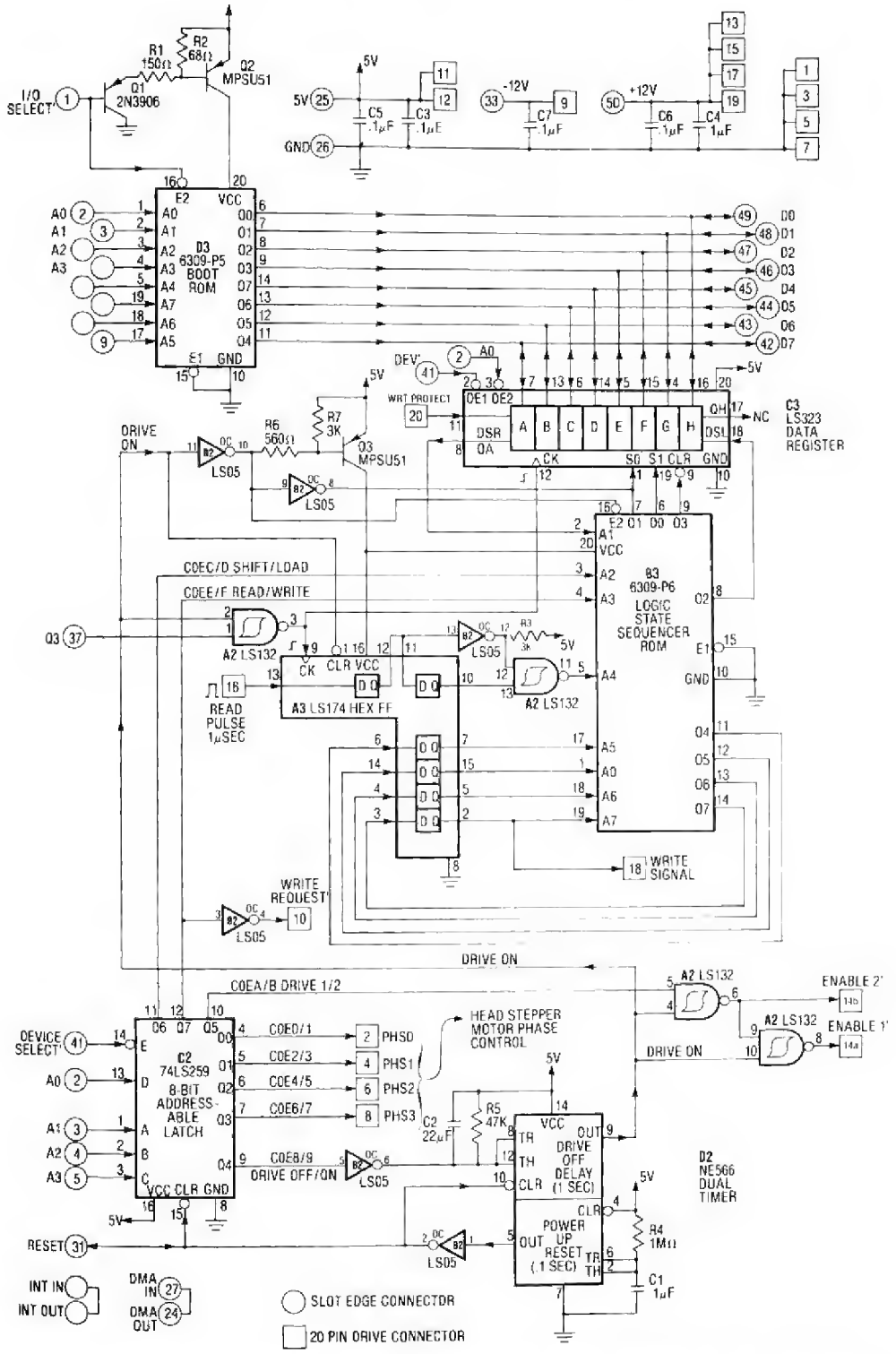


Figure 9.8 Schematic: The Disk II Controller (Address References Assume Slot 6).

bootstrap procedure can be continued. The Bootstrap ROM is connected to the address bus and data bus naturally, and its output enable is the I/O SELECT' input to the slot. This response to the I/O SELECT' input causes the DOS whose image resides on a drive 1 disk to be booted when PR#n or IN#n is executed from BASIC.*

The Autostart power-up RESET routine uses the contents of the Bootstrap ROM to detect the presence of a Disk II controller in the Apple. It does this by starting with Slot 7 and working downward, checking each slot for the presence of $20, $00, $03, and $3C at locations $Cn01, $Cn03, $Cn05, and $Cn07. When it finds this combination, it starts executing at $Cn00 on that slot, thus booting the DOS.

The Command Decoder

The overseeing 6502 program manages the controller via address bus commands in the DEVICE SELECT' range of the controller's slot ($C080,X through $C08F,X with slot number times $10 in the X-register). This range is divided up into eight off/on switches by an LS259 on the controller identically to the way the $C05X range on the motherboard is divided up (see TEXT, MIXED, PAGE2, HIRES, and AN0—AN3 in Figure 7.1). In other words, there are eight off/on soft switches by which a program can manage disk I/O. Like the motherboard screen mode switches and annunciators, the disk switches are set OFF by even addresses and ON by odd addresses. Table 9.1 is a brief listing of the functions of these address decoded commands, but more detailed explanations follow.

*See Chapter 7, PERIPHERAL SLOT CONNECTIONS and THE APPLE I/O SYSTEM: KSW AND CSW.

Drive Off/On and Drive Select

The $C088,X/$C089,X switch disables the drives or enables a drive, while the $C08A,X/$C08B,X switch selects drive 1 or drive 2 for enabling. Here is an illustrative, but otherwise useless program sequence:

```
LDX   #$60          SLOT 6
CMP   $C08B,X       SELECT DRIVE 2
CMP   $C089,X       DRIVE 2 ON
CMP   $C08A,X       DRIVE 2 OFF,
                    DRIVE 1 ON
CMP   $C088,X       DRIVES OFF
```

Turning a drive on ($C089,X) performs the following at the controller:

1. Applies +5V power to the sequencer ROM and the sequencer flip-flops at A3.
2. Applies Q3' to the clockpulse inputs of the data register and the sequencer control flip-flops. The sequencing and data transfer clock is Q3 falling. (DRIVES OFF forces the control flip-flops to clear.)
3. Enables the outputs of the sequencer ROM.
4. Enables sequencer control of the data register. (DRIVES OFF forces the data register to hold its present state.)
5. Causes the ENABLE 1' or the ENABLE 2' signal to go low depending on which drive is selected by the drive 1/drive 2 switch. At whichever drive is enabled, this turns on the drive motor and IN USE indicator, and it enables head positioning, writing, and control of the read pulse and write protect inputs to the controller.

Table 9.1 Disk II Controller Commands.

SWITCH	OFF FUNCTION	ON FUNCTION
Q0	$C080,X - PHASE 0 OFF	$C081,X - PHASE 0 ON
Q1	$C082,X - PHASE 1 OFF	$C083,X - PHASE 1 ON
Q2	$C084,X - PHASE 2 OFF	$C085,X - PHASE 2 ON
Q3	$C086,X - PHASE 3 OFF	$C087,X - PHASE 3 ON
Q4	$C088,X - DRIVES OFF	$C089,X - SELECTED DRIVE ON
Q5	$C08A,X - SELECT DRIVE 1	$C08B,X - SELECT DRIVE 2
Q6	$C08C,X - SHIFT WHILE WRITING, READ DATA	$C08D,X - LOAD WHILE WRITING, READ WRITE PROTECT
Q7	$C08E,X - READ	$C08F,X - WRITE

1. RESET' forces all switches off.
2. Access to even addresses causes the data register contents to be transferred to the data bus.

The drive off/on signal is routed through one half of an NE556 timer. The effect of this timer is to delay drive turn-off until one second after a reference to $C088,X. This gives the drive apparent momentum, keeping it running after it is turned off. The result is that the drive never turns off between closely spaced accesses, and in these instances, access time is reduced because there is no delay while the disk comes up to speed. This is why the computer is ready to process after a "CATALOG" well before the IN USE light goes out on the drive. The 1-second turn off delay does not apply to turning off the drive via a RESET. Pressing RESET causes the delay timer to clear and turns off the drive almost immediately.

The second half of the NE556 timer is used to generate a .1 second **power-up RESET**. This is necessary to achieve the disk autostart capibility on Revision 0 Apple II motherboards. These early Apples had no power-up reset generator. In the Apple IIe, the IOU generates a 35-millisecond power-up reset, so the only effect of the controller power-up reset circuitry is to extend the power-up reset to 100 milliseconds.

Head Positioning Commands

Address bus commands $C080,X through $C087,X are translated directly into four phase-off/phase-on stepper motor controls. These control voltages are routed to the drives where they are amplified and applied to the head positioning motor. Ascending references to $C080,X through $C087,X cause the head assembly on the enabled drive to move toward the inner track (track 34). Descending references cause movement toward track 0. The controlling program must wait approximately 20 milliseconds for motor response after turning a phase on. The actual motor response will vary with momentum, and the RWTS and DIIDD head positioning routines reduce positioning time by varying the waiting period with expected momentum.

The motor must be stepped **two phases per track**, so there are really 70 head positions. RWTS writes at the phase-0 aligned and the phase-2 aligned positions, but copy protected disks may have data written on the **half-tracks**, the phase-1 aligned and phase-3 aligned positions.* The head assembly has a **mechanical stop** (an electrical stop in some alternate drives) at track 0. One method to absolutely ascertain the head position is to step outward 80 phases as the bootstrap program does. The head will run against the track 0 stop, and you will be at track 0. The head/stepper motor linkage is aligned so that the motor will be **phase-0 aligned at track 0**, so from track 0 it is known that stepping inward must be begun by turning phase-1 on by a reference to $C083,X. From this point, the head position can be stored in RAM, and the phase alignment can be determined from the head position. After head positioning, the four phases should all be turned off, because the drive will behave as if a write protected disk is installed if phase-1 is left on.

*Another copy protection scheme is to turn two adjacent phases on then let the stepper motor settle approximately midway between them. This is referred to as stepping to a quarter track. The two phases must be turned off in rapid sequence if one of them is phase 1, the drive is unmodified, and data is to be written on the quarter track.

READ/WRITE

$C08E,X/$C08F,X is the controller's READ/WRITE soft switch. It is an addressing input to the sequencer and divides the sequencer into its two most significant parts, the **READ sequence** and the **WRITE sequence**. It also is inverted to become the WRITE REQUEST' signal to the drives. Thus, the READ/WRITE switch configures the controller for reading or writing via sequencer addressing, and it configures the enabled drive for writing via the WRITE REQUEST' line unless a write protected disk is installed.

SHIFT/LOAD

SHIFT/LOAD is a fairly inadequate label for the $C08C,X/$C08D,X soft switch, but any label would be. It is chosen because, during writing, $C08C,X causes shifting of the data register on every eighth sequencer clock, and $C08D,X causes loading of the data register from the data bus on every eighth sequencer clock. In reality, the SHIFT/LOAD switch performs several functions which defy summarization in a short label.

SHIFT/LOAD is an addressing input to the sequencer and it divides both the READ sequence and the WRITE sequence into two parts. As mentioned above, it is a programmable SHIFT/LOAD control for the data register. When the READ/WRITE switch is low, the SHIFT/LOAD function is changed to READ/CHECK WRITE PROTECT. These sequencer control functions are summarized in Table 9.2. The operation should become clearer when we study the sequencer listings.

$C08C,X and $C08D,X are also the normal input and output port addresses used by RWTS for transfer of disk data. In reality any even address could be used to load data from the data register to the MPU.

Table 9.2 Functions of the $C08C,X/$C08D,X and $C08E/$C08F Switches.

SHIFT/LOAD	READ/WRITE	Sequencer Function
$C08C,X	$C08F,X	Data register shift every eighth sequencer clock while writing.
$C08D,X	$C08F,X	Data register load every eighth sequencer clock while writing.
$C08C,X	$C08E,X	Enable READ sequencing.
$C08D,X	$C08E,X	Check state of write protect switch and initialize sequencer for writing.

although $C088 (DRIVES OFF) and $C08A (SELECT DRIVE 1) would be inappropriate for this purpose. Use of $C08D,X as the output address goes hand in hand with the fact that it causes loading of the data register from the data bus.

The Logic State Sequencer and Data Register

As mentioned before, the **logic state sequencer** is a ROM wired up to behave like a little 2 MHz computer. It is this powerful sequencer that enables the controller to perform such complex control with so few chips. It uses a 256-byte ROM with four of its data outputs (O4—O7) connected through flip-flops to four of its address inputs (A5, A0, A6, and A7). The flip-flops are clocked by **Q3 falling** while a drive is enabled.*

There are eight address inputs to the sequencer ROM, so let's refer to the four flip-flop latched end around inputs as the **sequencing inputs** and the other four address bits as the **partitioning inputs**. Assume for a moment that the four partitioning inputs are fixed. The overall ROM address will then change every time Q3 falls, and the contents of bits O4—O7 of any addressed location will determine the address after the next clockpulse. These four data bits thus contain flow information, and they are programmed so the flow will proceed in an orderly manner from clock to clock. There are 16 states of the four sequencing address inputs and 16 states of the four partitioning address inputs. The sequencer ROM is thus divided into 16 partitions of 16 sequencer states each.

The four partitioning inputs are:

A1 —QA, the MSB of the data register
A2 —SHIFT/LOAD, the $C08C,X/$C08D,X switch
A3 —READ/WRITE, the $C08E,X/$C08F,X switch
A4 —The read pulse from the disk drive

The A2 and A3 inputs allow the 6502 programmer to configure the sequencer for loading while writing, shifting while writing, reading data from the disk, or checking the write protect switch. The A1 and A4 inputs allow the sequencer flow to deviate depending on the presence or absence of a read pulse and the state of shifted data in the data register.

The **data register** is a very versatile 8-bit IC that can shift left, shift right, load parallel or store parallel based on its control inputs. The remaining four outputs of the sequencer ROM are connected to inputs of the data register, completing our picture of the sequencer. Four of the ROM data bits are programmed to control sequencing and the other four data bits are programmed to control the data register. There are only six distinct commands which the sequencer can cause the data register to perform, but there are 16 possible states of the command bits, due to redundant states which command the data register to do the same thing. Table 9.3 shows the 16-bit states in hexadecimal and binary with an asterisk next to the six states used by Apple to perform commands in their DOS 3.2 and 3.3 ROMs.

In addition to the sequencer control of the data register, any reference to even addresses in the DEVICE SELECT' range of the controller slot will cause the contents of the data register to be placed on the data bus. For this reason, programs should not cause the MPU to write to even addresses or the bidirectional peripheral data bus driver will compete with the data register for control of the peripheral data bus. It is important to note that, while a 6502 program can check the state of the data register at any time, a program can store data to the data register only when the sequencer is performing a load. As a result, the write operation involves getting the 6502 program in sync with the sequencer

*The sequencer clock is actually the rising edge of Q3', developed by gating Q3 through a 74LS132 NAND gate (see Figure 9.8). Due to propagation delay, the clock is effectively Q3 falling plus approximately 15 nanoseconds.

and keeping it there by performing write operations in exact timing loops. If the data register is to accept data from the MPU, the 6502 program must store the data to the data register at **exact multiples of the bit writing interval**. This interval is eight cycles of the sequencer or **four cycles of the MPU**.

It is important to make a distinction here between what **can** be done and what normally **is** done. A program can store data at any multiple of the bit writing interval—8, 12, 16, 20, etc. MPU cycles—and the data register will accept it. However, RWTS and DIIDD only store data to the data register at 32-, 36-, and 40-cycle intervals. There are very practical reasons for this which will become apparent as this discussion progresses.

Both the sequencer control flip-flops and the data register are clocked by Q3 falling when a drive is enabled. This means the sequencer operates at approximately 2 MHz, twice the frequency of the 6502. Additionally, the read pulse from the enabled drive is synchronized to the Q3 falling clock, quantized in pulse width to one Q3 period, and inverted in the process to form a negative pulse. It is this synchronized, negative read pulse which is applied to A4 of the sequencer ROM, and the read pulse is therefore monitored in the same 500-nanosecond time frame as the other addressing inputs to the ROM. In a further attempt to stabilize the shaky read pulse, a Schmidt trigger type NAND gate is used in the read pulse quantizing circuit (see Figure 9.8). Use of this type of gate increases noise immunity and ensures smooth transitions of the A4 input to the sequencer ROM.

During both reading and writing, the data register is **shifted left** while its most significant bit, QA, is monitored. In writing, the state of QA is monitored and the WRITE SIGNAL is toggled at the bit writing interval when QA is set. In reading, ONEs and ZEROs are shifted into the data register depending on the presence or absence of a read pulse at the bit writing interval. When QA becomes set, the sequencer holds the data register long enough for a 6502 program to detect the valid byte with a seven MPU cycle polling loop. Then the data register is cleared and the next byte is shifted in from the disk.

Decoding the Contents of the Sequencer ROM

In the past, the contents of the logic state sequencer ROM have been a mystery in the world of Apple users. The basic reason for this is that no one who understood the contents ever bothered to publish any information about it. A primary aim of this chapter is to fill this gap in Apple literature by providing formatted listings of the DOS 3.2 and DOS 3.3 sequencer ROM* contents and discussing operational aspects of disk I/O which are determined by the contents.

This section shows how to make the contents of the sequencer ROM accessible to the MPU and provides

*The logic state sequencer ROM was changed when Apple upgraded from DOS 3.2 to DOS 3.3 so the two versions of the ROM are referred to here as the DOS 3.2 and DOS 3.3 versions. The DOS 3.3 version has slightly different read sequences which improve the read reliability in the 16-sector formats of DOS 3.3, Pascal, and ProDOS.

Table 9.3 Logic State Sequencer Commands.

03-02-01-00			
HEX	BINARY	MNEMONIC	FUNCTION
*0	0000	CLR	CLEAR DATA REGISTER
1	0001	CLR	
2	0010	CLR	
3	0011	CLR	
4	0100	CLR	
5	0101	CLR	
6	0110	CLR	
7	0111	CLR	
*8	1000	NOP	NO OPERATION
*9	1001	SL0	SHIFT ZERO LEFT INTO DATA REGISTER
*A	1010	SR	SHIFT WRITE PROTECT SIGNAL RIGHT INTO DATA REGISTER
*B	1011	LD	LOAD DATA REGISTER FROM DATA BUS
C	1100	NOP	
*D	1101	SL1	SHIFT ONE LEFT INTO DATA REGISTER
E	1110	SR	
F	1111	LD	

a program which makes formatted listings of those contents. The program will accurately list the contents of any ROM designed to operate in the B3 socket of the Disk II controller. It is not necessary for the reader to go through the exercise of making listings of the DOS 3.2 and DOS 3.3 sequences since these listings are furnished in Figures 9.10 and 9.11. He may, however, find it interesting and educational to make his own listings.

The MPU cannot normally read the contents of the sequencer ROM because the ROM is not connected to the data bus. You can change this situation by removing the bootstrap ROM from the D3 socket and moving the sequencer ROM from the B3 socket to the D3 socket. The contents of the sequencer ROM can then be read by the MPU by addressing $C6XX (assumes Slot 6). Of course, your disk drive won't work with the controller configured this way, but you can save the data to cassette tape and then transfer it to a disk file when your controller is back to normal.

This data can be listed to a printer using the monitor, but it is not particularly readable in this form. I have written a BASIC program to format the data into a readable listing. Figures 9.9 through 9.11 contain listings of the Applesoft BASIC formatting program, the DOS 3.2 sequencer, and the DOS 3.3 sequencer respectively. The program was written in BASIC rather than assembly language so it could be more easily understood by readers who choose to study it. Also, readers should find it very easy to manipulate the sequencer listing because it is formatted as a 16 by 16 BASIC subscripted string variable. The program takes a little over a minute to run. It will list any sequencer ROM designed to run in the Disk II controller as long as its source file is read from the D3 socket of the controller. The source file needs to be resident on diskette as a binary file named "SEQROM". However, it would be easy to change line 50 of the program so the source file can be obtained elsewhere. Before running the program, enable the 80-column display or an 80-column printer.

The manipulations of the program are based on the following features of controller wiring:

1. Address bus A5 is connected to the A7 input to the D3 socket. Address bus A7 is connected to the A5 input to the D3 socket.
2. Data bus lines D4 through D7 are connected to outputs O7 through O4 on the D3 socket in reverse order.
3. A natural significance order of addressing inputs to the sequencer ROM is WRITE—READ PULSE—SHIFT/LOAD—QA—O7—O6—O5—O4. This does not correspond to the way they are connected to A7 through A0 on the sequencer ROM.
4. The read pulse applied to A4 of the sequencer ROM is a negative pulse.

These wiring connections were not designed to confuse us, although they serve the purpose. They were designed to minimize wire crossover on the mechanical layout. A design engineer can swap the address and data pin assignments on a ROM until he finds his own version of peace of mind, then he compensates for the scrambled pin assignments when he writes the ROM program.* The sequencer ROM formatting program must account for this by reading each byte of data, unscrambling address bits to find where that data is in the sequence, and reversing data bits before storing them in a 16 x 16 string variable matrix from which listings can easily be made.

The result is the listings of Figures 9.10 and 9.11.** The WRITE and READ sequences are separated from each other, and the listing is otherwise divided into columns of 16 sequencer states. This is a natural division since 16 different values can be represented by the four sequencing data bits. The far left hand column is the sequencer state, and the other columns hold the contents of the ROM for that sequencer state. **The left digit of each number is the next sequencer state, and the right digit is the command.** Next to each number is a mnemonic for the command digit.

*Steve Wozniak designed the Disk II controller. Working with Randy Wigginton, Wozniak completed this complicated and innovative design in the space of one week (the final week of 1977). Steve is very proud of the mechanical layout and at one time redid the layout for the sole purpose of reducing the number of feed-through holes from three to two. I had long assumed that much of Apple's disk interface technology had been borrowed from Shugart, but I couldn't have been further from the truth. The format and circuitry represent a notable creative effort by the Apple group. Besides the controller design, Wozniak wrote RWTS. Randy Wigginton wrote the rest of the original DOS, and Rod Holt designed the analog board of the disk drive.

**The formatted listings presented here are my own representations which resulted from lengthy investigation. The mnemonics, address labels, and layout won't, therefore, be the same as those used by Apple engineers. Who knows what they use? I submit, though, that my representation provides adequate illustration in the absence of any labels and formats published by Apple.

```
10  REM
12  REM              LIST LOGIC STATE SEQUENCER ROM
13  REM
14  REM                        BY JIM SATHER
15  REM                           2/22/83
16  REM
30  REM  THIS PROGRAM FORMATS AND LISTS THE PROGRAM CONTAINED IN THE
32  REM  DISK II LOGIC STATE SEQUENCER.  BEFORE RUNNING THIS PROGRAM,
33  REM  YOU MUST CREATE A BINARY SOURCE FILE ON A DISK AND
34  REM  NAME IT "SEQROM".  THE SOURCE FILE IS CREATED BY PLACING
36  REM  THE B3 ROM IN THE D3 SLOT OF THE DISK CONTROLLER.  THE
38  REM  SOURCE FILE IS WRITTEN TO CASSETTE FROM $C600-$C6FF.  WHEN THE
40  REM  CONTROLLER IS RESTORED, THE CASSETTE FILE IS TRANSFERRED TO DISK.
42  REM
50  PRINT  CHR$ (4);"BLOAD SEQROM,A7936"
100  DIM LST(15,15): DIM ASWAP(7): DIM DFIX(15): DIM HEX$(15): DIM CMND$(15)
110  FOR X = 0 TO 7: READ ASWAP(X): NEXT
120  FOR X = 0 TO 15: READ DFIX(X): NEXT
130  FOR X = 0 TO 15: READ HEX$(X): NEXT
140  FOR X = 0 TO 15: READ CMND$(X): NEXT
150  GR : COLOR= 6: REM SOMETHING TO WATCH WHILE WAITING
200  REM
201  REM  FORMAT DATA INTO 16 X 16 MATRIX
202  REM
210  FOR COL = 0 TO 15: FOR ROW = 0 TO 15:WRK = ROW + 16 * COL
300  REM
301  REM  GET BINARY FORM OF WRK
302  REM
305  FOR X = 0 TO 7:A(X) = 0
310 WRK% = WRK / 2:WRK = WRK / 2: IF (WRK - WRK%) THEN A(X) = 1
320 WRK = WRK%: NEXT X
400  REM
401  REM  RECONSTRUCT DECIMAL WRK WITH ADDRESS BITS SWAPPED:
402  REM  A7-A6-A5-A4-A3-A2-A1-A0 ---> A0-A2-A3-A6-A7-A5-A4-A1
403  REM  THE DATA WILL THEN BE ADDRESSED BY THE WORD:
404  REM  WRITE-READ PULSE-SHIFT/LOAD-QA-07-06-05-04.
405  REM
408 POWEROF2 = 1
410  FOR X = 0 TO 7: IF A(ASWAP(X)) THEN WRK% = WRK% + POWEROF2
420 POWEROF2 = POWEROF2 * 2: NEXT
500  REM
502  REM  SWAP BITS D7-D4 OF DATA BEFORE SAVING, BECAUSE THESE
503  REM  BITS ARE SWAPPED ON THE D3 ROM.
504  REM
507 DTA =  PEEK (WRK% + 7936):HI =  INT (DTA / 16)
510 LST(COL,ROW) = 16 * (DFIX(HI) + DTA / 16 - HI)
520  PLOT ROW,COL: NEXT ROW: NEXT COL: TEXT
600  REM
601  REM
602  REM  ADDRESS SWAP TABLE
610  DATA   1,4,5,7,6,3,2,0
620  REM
630  REM  DATA SWAP TABLE
640  DATA  0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15
650  REM
660  REM  DECIMAL TO HEX CONVERSION TABLE
670  DATA  0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
680  REM
690  REM  SEQUENCER COMMAND TABLE
695  DATA  -CLR,-CLR,-CLR,-CLR,-CLR,-CLR,-CLR,-CLR
700  DATA  -NOP,-SL0,-SR ,-LD ,-NOP,-SL1,-SR ,-LD
```

Figure 9.9 BASIC Listing: List State Sequencer ROM (1 of 2).

```
800  REM
802  REM  AT THIS POINT IN THE PROGRAM, THE SEQUENCER LISTING
810  REM  RESIDES IN 16 COLUMNS OF 16 SEQUENCES.  THESE COLUMNS
820  REM  ARE ARRANGED SO EACH COLUMN CONTAINS THE COMPLETE SEQUENCE
830  REM  FOR A STATE OF THE ADDRESS WORD:
840  REM  WRITE-READ PULSE-SHIFT/LOAD-QA.
900  REM
901  REM  *********      LIST WRITE SEQUENCE
902  REM
910  GOSUB 2000: GOSUB 2100: GOSUB 2200: GOSUB 2300: REM  PRINT WRITE HEADING.
920  FOR ROW = 0 TO 15: PRINT HEX$(ROW);"-";
930  FOR COL = 8 TO 15
940 HI% = LST(COL,ROW) / 16:LO% = (LST(COL,ROW) / 16 - HI%) * 16
950  PRINT "   ";HEX$(HI%);HEX$(LO%);CMND$(LO%);
960  NEXT COL: PRINT : NEXT ROW
999  GET B$: PRINT : PRINT
1000  REM
1001  REM  *********      LIST READ SEQUENCE
1002  REM
1010  REM  THE READ PROGRAM IS EASIER TO UNDERSTAND WHEN THE
1020  REM  LOAD PORTION IS SEPARATED FROM THE SHIFT PORTION AND
1024  REM  WHEN QA' (QA LOW) IS SEPERATED FROM QA.  THEREFORE
1030  REM  THE COLUMN ORDER IS CHANGED IN THE READ LISTING.
1040  REM
1045  REM  COLUMN ORDER DATA FOR READ LISTING
1050  DATA  0,4,1,5,2,6,3,7
1055  IF FLAG THEN 1070
1060  FOR X = 0 TO 7: READ CSWAP(X): NEXT :FLAG = 1
1070  REM
1080  GOSUB 2400: GOSUB 2500: GOSUB 2600: GOSUB 2700: REM PRINT READ HEADING.
1090  FOR ROW = 0 TO 15: PRINT HEX$(ROW);"-";
1100  FOR COL = 0 TO 7
1110 HI% = LST(CSWAP(COL),ROW) / 16:LO% = (LST(CSWAP(COL),ROW) / 16 - HI%) * 16
1120  PRINT "   ";HEX$(HI%);HEX$(LO%);CMND$(LO%);
1130  NEXT COL: PRINT : NEXT ROW: PRINT
1160  GET B$: PRINT : PRINT
1170  PRINT "CODE MNEMONIC  FUNCTION"
1180  PRINT " 0     CLR     CLEAR DATA REGISTER"
1190  PRINT " 8     NOP     NO OPERATION"
1200  PRINT " 9     SL0     SHIFT ZERO LEFT INTO DATA REGISTER"
1210  PRINT " A     SR      SHIFT WRITE PROTECT SIGNAL RIGHT INTO DATA REGISTER"
1220  PRINT " B     LD      LOAD DATA REGISTER FROM DATA BUS"
1230  PRINT " D     SL1     SHIFT ONE LEFT INTO DATA REGISTER"
1240  END
1990  REM
2000  PRINT : PRINT ; SPC( 37);"WRITE": PRINT : RETURN
2100  PRINT "    *----------READ PULSE----------*----------NO READ PULSE----------*"
2110  RETURN
2200  PRINT "    *------SHIFT-------*-------LOAD--------*-------SHIFT-------*-------LOAD-------*"
2210  RETURN
2300  PRINT "SEQ *--QA'--*---QA---*--QA'---*---QA---*--QA'---*---QA---*--QA'---*---QA--*"
2310  PRINT : RETURN
2400  PRINT ; SPC( 37);"READ": PRINT : RETURN
2500  PRINT "    *--------------SHIFT---------------*---------------LOAD---------------*"
2510  RETURN
2600  PRINT "    *-------QA'--------*--------QA---------*--------QA'--------*--------QA--------*"
2610  RETURN
2700  PRINT "SEQ *--RP---*-NO RP--*---RP---*-NO RP--*---RP---*-NO RP--*---RP---*-NO RP-*"
2710  PRINT : RETURN
```

Figure 9.9 BASIC Listing: List State Sequencer ROM (2 of 2).

WRITE

	READ PULSE				NO READ PULSE			
	SHIFT		LOAD		SHIFT		LOAD	
SEQ	QA'	QA	QA'	QA	QA'	QA	QA'	QA
0-	18-NOP	18-NOP	18-NOP	18-NOP	18-NOP	18-NOP	18-NOP	18-NOP
1-	28-NOP	28-NOP	28-NOP	28-NOP	28-NOP	28-NOP	28-NOP	28-NOP
2-	39-SL0	39-SL0	3B-LD	3B-LD	39-SL0	39-SL0	3B-LD	3B-LD
3-	48-NOP	48-NOP	48-NOP	48-NOP	48-NOP	48-NOP	48-NOP	48-NOP
4-	58-NOP	58-NOP	58-NOP	58-NOP	58-NOP	58-NOP	58-NOP	58-NOP
5-	68-NOP	68-NOP	68-NOP	68-NOP	68-NOP	68-NOP	68-NOP	68-NOP
6-	78-NOP	78-NOP	78-NOP	78-NOP	78-NOP	78-NOP	78-NOP	78-NOP
7-	08-NOP	88-NOP	08-NOP	88-NOP	08-NOP	88-NOP	08-NOP	88-NOP
8-	98-NOP	98-NOP	98-NOP	98-NOP	98-NOP	98-NOP	98-NOP	98-NOP
9-	A8-NOP	A8-NOP	A8-NOP	A8-NOP	A8-NOP	A8-NOP	A8-NOP	A8-NOP
A-	B9-SL0	B9-SL0	BB-LD	BB-LD	B9-SL0	B9-SL0	BB-LD	BB-LD
B-	C8-NOP	C8-NOP	C8-NOP	C8-NOP	C8-NOP	C8-NOP	C8-NOP	C8-NOP
C-	D8-NOP	D8-NOP	D8-NOP	D8-NOP	D8-NOP	D8-NOP	D8-NOP	D8-NOP
D-	E8-NOP	E8-NOP	E8-NOP	E8-NOP	E8-NOP	E8-NOP	E8-NOP	E8-NOP
E-	F8-NOP	F8-NOP	F8-NOP	F8-NOP	F8-NOP	F8-NOP	F8-NOP	F8-NOP
F-	88-NOP	08-NOP	88-NOP	08-NOP	88-NOP	08-NOP	88-NOP	08-NOP

READ

	SHIFT				LOAD			
	QA'		QA		QA'		QA	
SEQ	RP	NO RP	RP	NO RP	RP	NO RP	RP	NO RP
0-	D8-NOP	18-NOP	18-NOP	08-NOP	0A-SR	0A-SR	0A-SR	0A-SR
1-	D8-NOP	28-NOP	28-NOP	28-NOP	0A-SR	0A-SR	0A-SR	0A-SR
2-	D8-NOP	38-NOP	38-NOP	38-NOP	0A-SR	0A-SR	0A-SR	0A-SR
3-	D8-NOP	48-NOP	D8-NOP	48-NOP	0A-SR	0A-SR	0A-SR	0A-SR
4-	D8-NOP	58-NOP	D8-NOP	58-NOP	0A-SR	0A-SR	0A-SR	0A-SR
5-	D8-NOP	68-NOP	D8-NOP	68-NOP	0A-SR	0A-SR	0A-SR	0A-SR
6-	D8-NOP	78-NOP	D8-NOP	78-NOP	0A-SR	0A-SR	0A-SR	0A-SR
7-	D8-NOP	88-NOP	D8-NOP	88-NOP	0A-SR	0A-SR	0A-SR	0A-SR
8-	D8-NOP	98-NOP	D8-NOP	98-NOP	0A-SR	0A-SR	0A-SR	0A-SR
9-	D8-NOP	09-SL0	D8-NOP	A8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
A-	CD-SL1	BD-SL1	D8-NOP	B8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
B-	D9-SL0	39-SL0	D8-NOP	C8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
C-	D9-SL0	D9-SL0	D8-NOP	A0-CLR	0A-SR	0A-SR	0A-SR	0A-SR
D-	1D-SL1	0D-SL1	E8-NOP	E8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
E-	FD-SL1	FD-SL1	F8-NOP	F8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
F-	DD-SL1	4D-SL1	E0-CLR	E0-CLR	0A-SR	0A-SR	0A-SR	0A-SR

CODE	MNEMONIC	FUNCTION
0	CLR	CLEAR DATA REGISTER
8	NOP	NO OPERATION
9	SL0	SHIFT ZERO LEFT INTO DATA REGISTER
A	SR	SHIFT WRITE PROTECT SIGNAL RIGHT INTO DATA REGISTER
B	LD	LOAD DATA REGISTER FROM DATA BUS
D	SL1	SHIFT ONE LEFT INTO DATA REGISTER

Figure 9.10 The DOS 3.2 Logic State Sequencer.

WRITE

	READ PULSE				NO READ PULSE			
	SHIFT		LOAD		SHIFT		LOAD	
SEQ	QA'	QA	QA'	QA	QA'	QA	QA'	QA
0-	18-NOP	18-NOP	18-NOP	18-NOP	18-NOP	18-NOP	18-NOP	18-NOP
1-	28-NOP	28-NOP	28-NOP	28-NOP	28-NOP	28-NOP	28-NOP	28-NOP
2-	39-SL0	39-SL0	3B-LD	3B-LD	39-SL0	39-SL0	3B-LD	3B-LD
3-	48-NOP	48-NOP	48-NOP	48-NOP	48-NOP	48-NOP	48-NOP	48-NOP
4-	58-NOP	58-NOP	58-NOP	58-NOP	58-NOP	58-NOP	58-NOP	58-NOP
5-	68-NOP	68-NOP	68-NOP	68-NOP	68-NOP	68-NOP	68-NOP	68-NOP
6-	78-NOP	78-NOP	78-NOP	78-NOP	78-NOP	78-NOP	78-NOP	78-NOP
7-	08-NOP	88-NOP	08-NOP	88-NOP	08-NOP	88-NOP	08-NOP	88-NOP
8-	98-NOP	98-NOP	98-NOP	98-NOP	98-NOP	98-NOP	98-NOP	98-NOP
9-	A8-NOP	A8-NOP	A8-NOP	A8-NOP	A8-NOP	A8-NOP	A8-NOP	A8-NOP
A-	B9-SL0	B9-SL0	BB-LD	BB-LD	B9-SL0	B9-SL0	BB-LD	BB-LD
B-	C8-NOP	C8-NOP	C8-NOP	C8-NOP	C8-NOP	C8-NOP	C8-NOP	C8-NOP
C-	D8-NOP	D8-NOP	D8-NOP	D8-NOP	D8-NOP	D8-NOP	D8-NOP	D8-NOP
D-	E8-NOP	E8-NOP	E8-NOP	E8-NOP	E8-NOP	E8-NOP	E8-NOP	E8-NOP
E-	F8-NOP	F8-NOP	F8-NOP	F8-NOP	F8-NOP	F8-NOP	F8-NOP	F8-NOP
F-	88-NOP	08-NOP	88-NOP	08-NOP	88-NOP	08-NOP	88-NOP	08-NOP

READ

	SHIFT				LOAD			
	QA'		QA		QA'		QA	
SEQ	RP	NO RP	RP	NO RP	RP	NO RP	RP	NO RP
0-	18-NOP	18-NOP	18-NOP	18-NOP	0A-SR	0A-SR	0A-SR	0A-SR
1-	2D-SL1	2D-SL1	38-NOP	38-NOP	0A-SR	0A-SR	0A-SR	0A-SR
2-	D8-NOP	38-NOP	08-NOP	28-NOP	0A-SR	0A-SR	0A-SR	0A-SR
3-	D8-NOP	48-NOP	48-NOP	48-NOP	0A-SR	0A-SR	0A-SR	0A-SR
4-	D8-NOP	58-NOP	D8-NOP	58-NOP	0A-SR	0A-SR	0A-SR	0A-SR
5-	D8-NOP	68-NOP	D8-NOP	68-NOP	0A-SR	0A-SR	0A-SR	0A-SR
6-	D8-NOP	78-NOP	D8-NOP	78-NOP	0A-SR	0A-SR	0A-SR	0A-SR
7-	D8-NOP	88-NOP	D8-NOP	88-NOP	0A-SR	0A-SR	0A-SR	0A-SR
8-	D8-NOP	98-NOP	D8-NOP	98-NOP	0A-SR	0A-SR	0A-SR	0A-SR
9-	D8-NOP	29-SL0	D8-NOP	A8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
A-	CD-SL1	BD-SL1	D8-NOP	B8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
B-	D9-SL0	59-SL0	D8-NOP	C8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
C-	D9-SL0	D9-SL0	D8-NOP	A0-CLR	0A-SR	0A-SR	0A-SR	0A-SR
D-	D8-NOP	08-NOP	E8-NOP	E8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
E-	FD-SL1	FD-SL1	F8-NOP	F8-NOP	0A-SR	0A-SR	0A-SR	0A-SR
F-	DD-SL1	4D-SL1	E0-CLR	E0-CLR	0A-SR	0A-SR	0A-SR	0A-SR

CODE	MNEMONIC	FUNCTION
0	CLR	CLEAR DATA REGISTER
8	NOP	NO OPERATION
9	SL0	SHIFT ZERO LEFT INTO DATA REGISTER
A	SR	SHIFT WRITE PROTECT SIGNAL RIGHT INTO DATA REGISTER
B	LD	LOAD DATA REGISTER FROM DATA BUS
D	SL1	SHIFT ONE LEFT INTO DATA REGISTER

Figure 9.11 The DOS 3.3 Logic State Sequencer.

As a quick example of interpreting this listing, assume that the sequencer is at the top of the fifth column of the WRITE sequence in Figure 9.10. This means the sequencer is at State 0 and the partitioning address bits are WRITE—NO READ PULSE—SHIFT—QA'. The data being driven out of the sequencer ROM is 18. **The 1 is the next sequencer state** which will occur when Q3 falls. **The 8 is the command which will be executed** when Q3 falls, a NOP. A quick scan through the WRITE listing will show that it contains mostly NOPs and that it normally flows from one sequencer state to the next.

The WRITE Sequence

The WRITE sequence and WRITE PROTECT sequence are the same in the 3.2 and 3.3 sequencers, so this section pertains to both. Please refer to the 3.3 listing during these discussions, using the 3.2 listing for comparison.

We begin our analysis by making two simplifying observations. First, if WRITE is selected at the READ/WRITE switch ($C08F,X), the left four columns (READ PULSE) are identical to the right four columns (NO READ PULSE). This means that read data has no effect on the WRITE sequence and it can be ignored for purposes of writing. This is a necessity because meaningless read pulses will normally be present while writing. Second, all entries of the READ—LOAD sequence states are 0A. This means that READ—LOAD sets the sequencer state to 0 and idly shifts the state of the write protect switch **right** into the data register where it can by checked by a 6502 program. Thus we have the basis for checking if a disk is write protected, namely:

```
LDA   $CØ8D,X     CHECK WRITE
                  PROTECT IF READ.
LDA   $CØ8E,X     READ.
BMI   ERROR       BRANCH TO WRITE
                  PROTECT ROUTINE
                  IF QA SET.
```

These are the program steps used by RWTS to check for write protection before writing the data block of a sector. A routine such as this must be performed every time before writing. It not only checks if a disk is write protected, it also clears QA if the disk is not write protected and sets the sequencer to State 0. This initializes the sequencer for writing, and it is the only way an MPU program can establish the state of the sequencer.

In the above program steps, the data register was checked via a LDA $C08E,X. This sets READ/WRITE to READ, shifts the state of the write protect switch into QA of the data register, and places this shifted value on the data bus so the MPU can load it to its accumulator. Now, in reality, any time you run this program, the READ/WRITE switch will already be set to READ, because we are preparing to write. If READ/WRITE were set to WRITE and a disk with no write protect was rotating, the logic state sequencer would be merrily stomping the flux out of all the data on the track. When you write to the disk, the program waits until the right moment, switches READ/WRITE to WRITE, then stores a few hundred bytes of data to the data register in precision loops, then switches back to READ. So, since the READ/WRITE switch was already in READ above, the "LDA $C08E,X" is not intended to switch READ/WRITE to READ. It is intended solely to load the contents of the data register from the most appropriate even controller address $C08E,X.

If for some reason the above write protect check were entered with the READ/WRITE switch in WRITE, the write protect switch would still be read correctly. This is some pretty fast addressing, shifting, storing, and loading, the timing for which is illustrated in Figure 9.12. This figure shows the last PHASE 0 period of an MPU access to the READ/WRITE or SHIFT/LOAD switch. The whole operation depends on the very brief access time (70 nanoseconds) of the 6309 PROM that Apple uses for the sequencer ROM. This enables DEVICE SELECT' to fall, the accessed switch to change states, and data out of the sequencer ROM to become valid in plenty of time for the data register to be newly configured when the controller clock rises. The controller clock is Q3 falling or, more accurately, Q3 inverted rising with a typical propagation delay of 15 nanoseconds experienced in the inversion. The data register then responds to its clock well before PHASE 0 falls, allowing the MPU to load the new state of the data register. All this means that the following general rule applies: when the MPU reads the data register, the sequencer responds to any new configuration, performs a resulting operation on the data register, and then gates the contents of the data register to the data bus for reading by the MPU.

Now assume a 6502 program has checked the write protect switch and found it can now write to the disk. The program can then write a byte of data to the disk with these steps:

```
LDA   DATA1
STA   $CØ8F,X     WRITE
CMP   $CØ8C,X     SHIFT
```

Figure 9.13 shows the timing of what happens here. The "STA $C08F,X" without indexing across a page

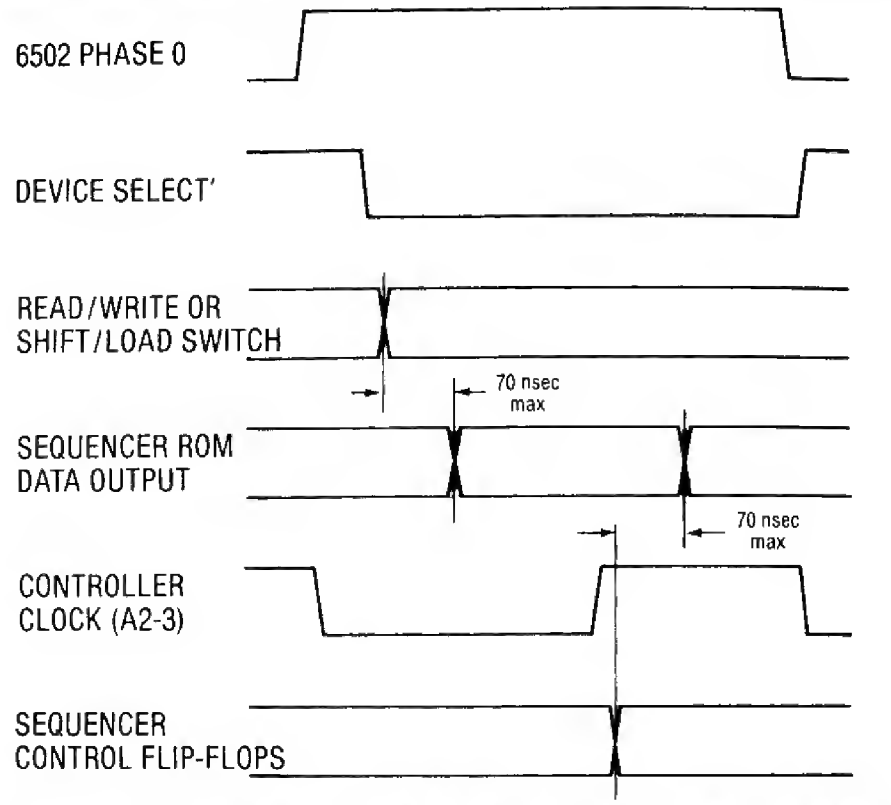


Figure 9.12 Timing Diagram: Changing the READ/WRITE or SHIFT/LOAD Switch.

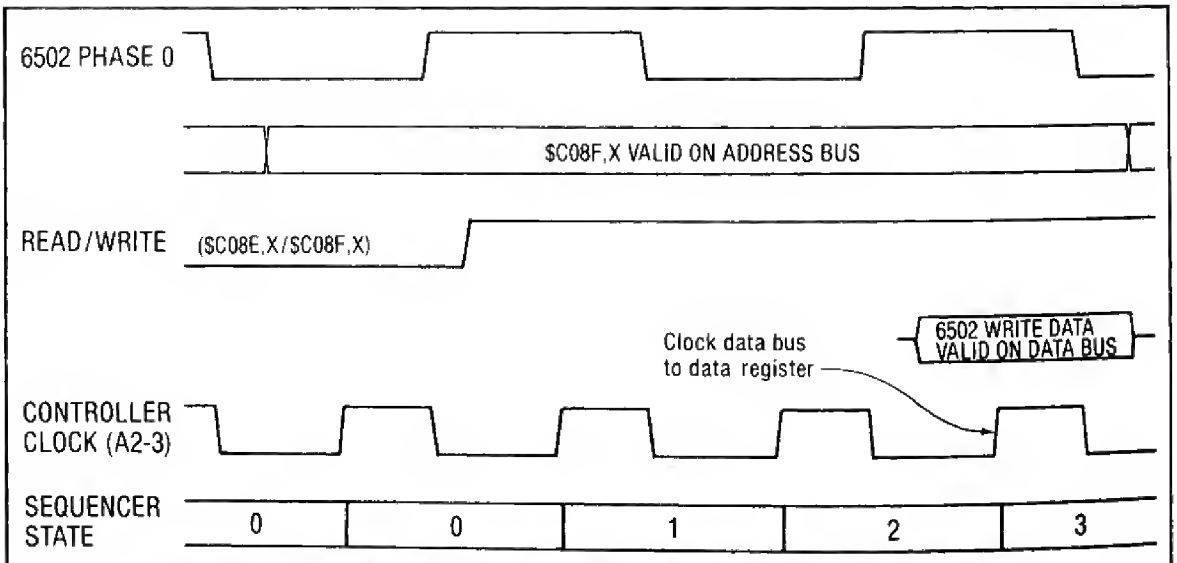


Figure 9.13 Timing Diagram: Switching to Write After Checking Write Protect.

boundary actually puts $C08F,X on the address bus for the last two MPU cycles of this 5-cycle instruction. The first $C08F,X cycle is a read access and the second is a write access with the MPU controlling the data bus during the greater part of PHASE 0. The first $C08F,X cycle causes READ/WRITE to switch to WRITE about 90 nanoseconds after PHASE 0 rises. Within 70 nanoseconds of this event, the sequencer ROM's data outputs have responded to the new address input. The sequencer state is still 0. That won't change until the sequencing flip-flops sense a clock edge. The sequencer is therefore waiting for a clock edge, sitting at State 0—WRITE—LOAD—QA'. There may or may not be a read pulse, but we don't care. Assume there is no read pulse.

Now look at the WRITE sequence in Figure 9.11 (finally). We are at the top of column seven where a 18-NOP is found. At the next sequencer clock nothing is done to the data register, but the sequencer moves up to State 1. State 1 contains a 28-NOP, which causes sequencing to State 2 at the next clock. State 2 contains a 3B-LD, which causes sequencing to State 3 at the next clock as well as the loading of the data register from the data bus.

Now look again at Figure 9.13. The sequencer clock edge in State 2 occurs while the MPU is placing valid data from the "STA $C08F,X" instruction on the data bus, and it therefore has succeeded in storing data to the data register. Note that a "STA $C0EF" (assumes Slot 6) would not have worked here because MPU data would have been on the data bus at State 0 instead of State 2. **Clearly, Apple designed the WRITE sequence to support the features of the 6502 "STA ABSOLUTE,X" instruction with no page crossing.**

Now look back at Figure 9.11. Assume that when the MPU stored data in the data register, it set QA. In fact, if the data was written by RWTS or DIIDD, it's a sure thing—**QA is set.** We will see that all data words that RWTS or DIIDD store to the disk have their MSB set. This means that instead of sequencing to State 3 in column 7, the sequencer goes to State 3 of column 8. It makes no difference here, but it will when we reach State 7. Until that point is reached nothing happens. At State 7, if QA is set, sequencing up the states continues, but if QA is reset, the sequencer loops back to State 0. Right now, QA is set, so the next state is State 8.

When State 8 is entered from State 7, the WRITE signal switches from low to high. Why? Because the WRITE signal is connected to the A7 input of the sequencer ROM. The sequencing address bits are A7—A6—A0—A5, so A7 is low in states 0—7 and high in states 8—F (see Figure 9.8). Now think about the decision that was made at State 7 in this light. **If QA was reset, the WRITE signal was left alone. If QA was set, the WRITE signal was toggled.**

Continuing on in column 8, State A is reached before the next event. Remember that "STA $C08F,X" was followed by "CMP $C08C,X". Just as we arrive at State A, this second instruction switches SHIFT/LOAD to SHIFT, causing us to arrive at State A in column 6 instead of 8. This is just in time to cause shifting instead of loading at State A, because there is a SL0 in column 6 compared to the LD in column 8. The "CMP $C08C,X" is barely short enough to meet the timing requirements for switching SHIFT/LOAD to SHIFT.

The SL0 causes the next bit to be shifted into QA, while QH, the LSB of the data register, is filled with a ZERO. SL0 is functionally similar to the 6502 instruction, ASL. From this point, we sequence up to State F in column 5 or 6 depending on whether QA was set or reset by the SL0. From State F, the sequencer will loop back to State 0 if QA is set, toggling the WRITE signal, or it will loop back to State 8, leaving the WRITE signal high, if QA is reset.

Now let's step back and look at what's happening. **Writing to the disk is a load and shift process**, a little like HIRES pattern outputs but much slower. Also, the MPU takes a very active role in the loading and shifting of disk write data. There are two 8-state loops in the WRITE sequence. After initializing the WRITE sequence, data is stored in the data register at a critical point in the A7' loop. As quickly thereafter as the 6502 can do it, the sequencer is configured to shift left at the critical point instead of loading. Then the MPU goes about its business while the sequencer continues looping, shifting the data register. **If the sequencer senses QA high, flow vectors to the opposite loop, toggling the WRITE signal.**

Figure 9.14 is a flowchart of the WRITE sequence, which may help you interpret the listing. The flowchart is reminiscent of the schematic of a flip-flop. This is not surprising because the WRITE sequence has functional similarities to a toggling flip-flop.

So, the sequencer outputs eight bits of data. What then? Well, if the program controlling the MPU did nothing more, after the eighth bit was shifted out, the data register would be all zeroes and no more field reversals would be written on the disk. This constant field condition cannot be read by the drives but results in sporadic read pulses. In other words, the MPU needs to stay involved. Normally, the MPU program will wait until the last bit has been shifted to the disk, then switch SHIFT/LOAD to

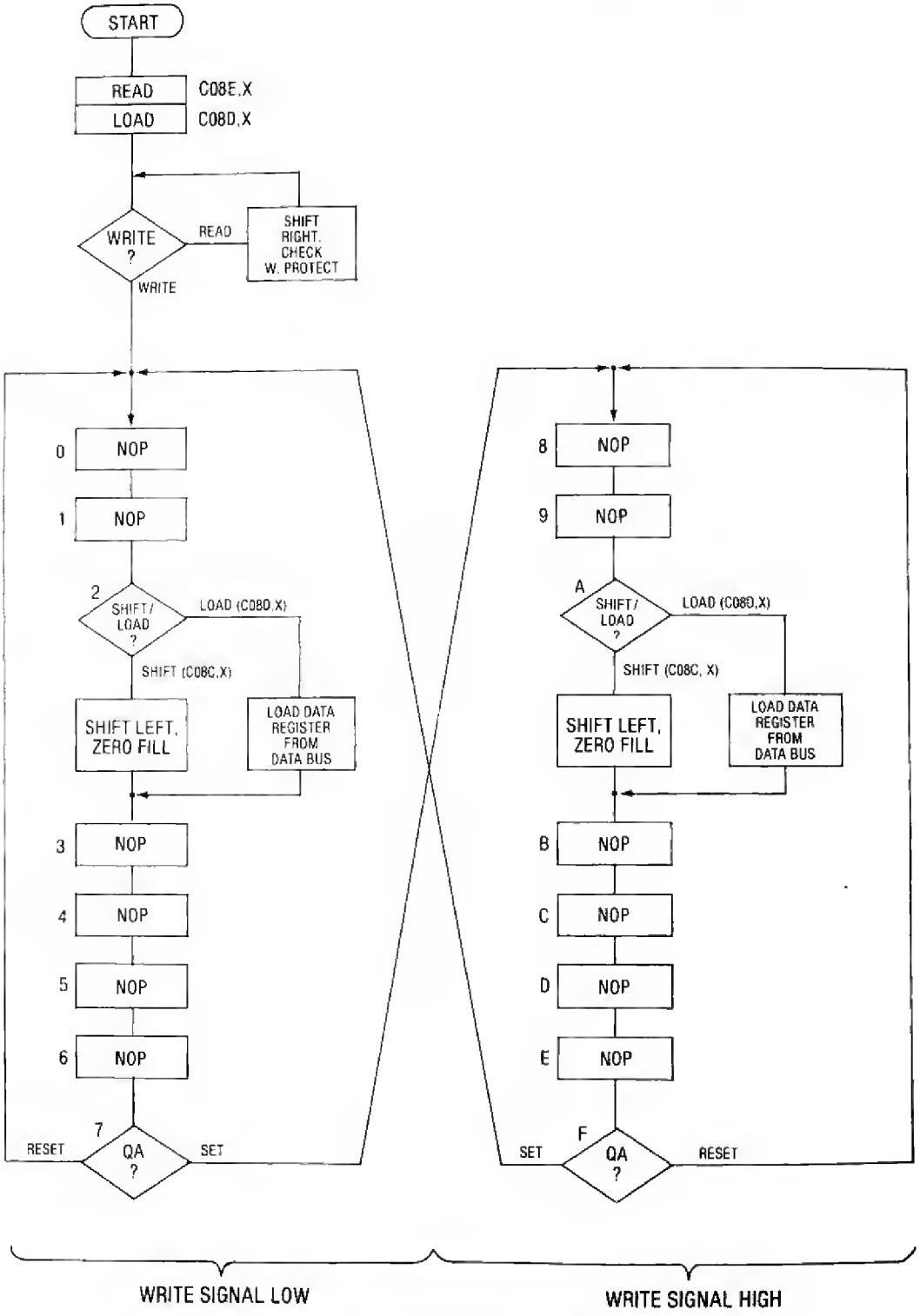


Figure 9.14 Flowchart of the Write Sequence.

LOAD with a STA instruction at the precise moment that the data register will accept it. This moment occurs once every sequencer write loop, once every eight sequencer clocks, or once every four MPU cycles. It takes 32 MPU cycles to output eight bits of information, so **to continuously shift out information in 8-bit groups, the MPU must store data at $C08D,X every 32 cycles, then immediately enable shifting with a read access to $C08C,X.** This example writes a byte to the disk then begins writing a second byte.

```
LDA   $C08D,X     LOAD
LDA   $C08E,X     READ
BMI   ERROR       WRITE PROTECT ERR
LDA   DATA1
STA   $C08F,X     WRITE DATA1
CMP   $C08C,X     SHIFT (4CP)
PHA               (3CP)
PLA               (4CP)
PHA               (3CP)
PLA               (4CP)
BIT   $0          (3CP)
NOP               (2CP)
LDA   DATA2       (4CP)
STA   $C08D,X     LOAD (5CP):
                  TOTAL = 32CP
CMP   $C08C,X     SHIFT
```

The above example illustrates the principle of writing continuous bytes of data: initiate the WRITE sequence, then store data every 32 cycles. This will normally be accomplished in 32-cycle loops. After storing the last byte to be written, wait 32 cycles, switching READ/WRITE to READ on the 32nd cycle. You don't have to write in 32-cycle loops. You can store 6-bit data words in 24-cycle loops if you can figure out some purpose for it. You can store data in any multiple of 4 cycles and the data register will accept it. Read syncing leaders are written by storing $FF to the data register in 36-cycle loops (DOS 3.2) or 40-cycle loops (DOS 3.3 or DIIDD).* This creates a series of 111111110 or 1111111100 strings which, we will see, syncs up the sequencer for reading following data.

Disk Data Formats

There is an inherent problem with storing data on floppy disks. As their name suggests, floppies are less than rigid. This and other factors contribute to the following fact of life: just because some Apple wrote data to a disk at a 4-cycle bit interval doesn't mean that read pulses are going to come back from the disk at the same interval. In the first place, the Apple has a built-in clockpulse jitter with every 65th MPU cycle longer than the rest. This elongates some write intervals by 140 nanoseconds, and it doesn't help. But normal problems with the floppy medium and drive inconsistencies are more significant in causing read pulse variations.

Realizing that the read pulse varies a lot, consider how this variation affects trying to detect the absence of a pulse. In reading the absence of a pulse, the READ sequence must wait a certain amount of time past the last actual pulse, then decide "there was definitely no pulse there: that sure was a ZERO." In the absence of a clockpulse coming off the disk, the previous read pulse, or ONE, becomes the time reference for the absence of a read pulse, or a ZERO.

What if you have two ZEROs in a row? Well the last ONE bit is the time reference for the second and all succeeding ZEROs in a string. Suppose the reading drive rotates at 280 RPM but the disk was written using a drive that rotated at 320 RPM. You cannot read a long string of ZEROs under these conditions, because while the write interval was eight controller cycles, the read interval could be ten cycles. Furthermore, in the absence of field reversals on the disk beyond three or so write intervals, the read interface chip in the drive begins to generate invalid read pulses. You simply cannot read a long string of ZEROs, because your time reference is unstable and your read interface can't do it anyway.

How may ZEROs in a row can be read reliably? Two. RWTS 3.3 and DIIDD **never write more than two ZEROs in a row, and RWTS 3.2 never writes more than one ZERO in a row.** Of course, normal data bytes very often have more than two adjacent zero bits. These normal data bytes cannot be stored directly to the disk but must be translated to disk compatible 8-bit words. It takes more than one LOAD-SHIFT cycle to output a data byte, because 256 possible numbers can be represented in a normal byte, but not nearly as many can be represented if you place restrictions on the number of adjacent ZEROs.

There is a second restriction on data which the Apple stores to the disk. **The MSB is always set.** The MSB is used by both the READ sequence and 6502 programs to define when a byte of read data is in the data register. The MSB is therefore not data

*In some literature, the read syncing leaders are referred to as a series of autosync bytes or self-sync bytes.

but the **BYTE FLAG.*** It serves as a data gap which allows the READ sequencer to hold the previous byte in the data register for a long enough time that a 6502 program can detect the presence of a complete byte in a 7-cycle polling loop:

```
POLLIT    LDA $CØ8C,X
          BPL POLLIT
```

The Apple Disk II was first released with DOS 3 and the associated controller. Then the DOS was upgraded to 3.1, then 3.2, then 3.2.1 with no change in written data format. This format will be referred to here as the 3.2 format. The 3.2 data restrictions are MSB set and no two adjacent bits reset. All sector data are written using the following 32 values:

RWTS 3.2 WRITE TABLE

```
BC9A - AB AD AE AF B5 B6
BCAØ - B7 BA BB BD BE BF D6 D7
BCA8 - DA DB DD DE DF EA EB ED
BCBØ - EE EF F5 F6 F7 FA FB FD
BCB8 - FE FF
```

The values D5 and AA are also valid, and they are used as the first two identifying values which precede every address field and data field. A 5-bit word can contain 32 values, so the 3.2 writing process involves taking a 256-byte data block and translating it into 410 5-bit words. The 410 5-bit words do not directly index the write table to find the byte to be written. Rather, the first word written directly indexes the write table. The second index is an exclusive-OR between the first and second words: Each following index is an exclusive-OR between the current and previous word. At the end, a 411th word is written. The last word of the coded buffer directly indexes the write table for this word. When this process is reversed in the read operation, each byte has to be correctly read for the following bytes to be read correctly. The 411th word read serves as a checksum and must be equal to the 410th decoded word or the read will be deemed unsuccessful and revert to a try again loop. This check sum procedure is an effective method of verifying the validity of disk data transfer.

Apple increased the disk data density in the DOS 3.3 upgrade by easing the restriction on adjacent ZEROs. The 3.2 controller can read this data, but it's a struggle and the possibility of reading errors is great. Apple improved the read reliability by changing the sequencer ROM. They also made the questionable move of changing the Bootstrap ROM and bootstrap procedure, the most notable result of which is that a 3.3 controller will not boot a 3.2 disk. This incompatibility is due solely to program based conventions. The 3.3 controller is fully capable of reading anything written on 3.2 disks.

The DOS 3.3 restrictions on written data are MSB set, no more than one pair of adjacent ZEROs, and at least one pair of adjacent ONEs in bits 6 through 0. D5 and AA are still used only as field identifiers, and they don't meet the pair of adjacent ONEs requirements. This is notable because it helps distinguish D5 and AA from the other valid written words. The restriction requiring a pair of adjacent ONEs rules out 95, A5, A9, and CA. Besides D5 and AA, the DOS 3.3 written values are:

RWTS 3.3 WRITE TABLE

```
BA29 - 96 97 9A 9B 9D 9E 9F
BA3Ø - A6 A7 AB AC AD AE AF B2
BA38 - B3 B4 B5 B6 B7 B9 BA BB
BA4Ø - BC BD BE BF CB CD CE CF
BA48 - D3 D6 D7 D9 DA DB DC DD
BA5Ø - DE DF E5 E6 E7 E9 EA EB
BA58 - EC ED EE EF F2 F3 F4 F5
BA6Ø - F6 F7 F9 FA FB FC FD FE
BA68 - FF
```

There are 64 values in the write table, so you can guess that six bits are written per LOAD-SHIFT cycle, and a 256-byte block is written in 342 LOAD-SHIFT cycles. With the size of the data field in a sector thus reduced, Apple was able to increase the number of sectors per track from 13 to 16.

The most recent development in the evolution of Apple IIe disk I/O has been the release of ProDOS. No changes were required in the sequencer ROM to upgrade to ProDOS, and to a very great extent, ProDOS data formats are identical to those of DOS 3.3.* The same write table is used to encode data although the table format is different than that of RWTS 3.3 (see $FA00—$FAFF of DIIDD), and address and data fields are identified with the same series of bytes. Sectors on a ProDOS formatted disk can be read and written by RWTS 3.3, and pairs of sectors on a DOS 3.3 formatted disk can be read and written by DIIDD. There are some minor differences between DOS 3.3 and ProDOS formats (see

*Steve Wozniak originally devised the Apple disk formats. In a speech given in Anaheim, California, on April 17, 1983, he said that his idea for flagging groups of data by having every eighth bit set came directly from the "stop bit" used in RS232 data transmission.

*The equivalence of DOS 3.3 and ProDOS data formats extends only to data encoded on the disk at the track/sector level. As has been mentioned, file structure in the two operating systems is completely different, as is the treatment of data as 2-sector blocks by ProDOS.

Figure 9.15), but they do not affect operation and do not affect the following discussion. Therefore, only two data formats will be referred to—the DOS 3.2 format and the DOS 3.3/ProDOS format.

An important point concerning Apple disk data formats is that you can write any sort of bit stream you desire, but you must write something that can be read by the logic state sequencer. The sequencer was designed to read a certain data format, and it's all it can do to read this floppy data reliably. Copy protect artists must study the READ sequence very thoroughly to discover ways to write bit streams which can be synced by the READ sequence with some secret manipulation by a 6502 program.

We will see that the READ sequence will properly read streams of data written from the 3.2, 3.3, and ProDOS write tables in 32-cycle loops. It does take an indefinite period of time for the sequencer to sync up when it first encounters a random stream of data. However, random data streams in the DOS formats are always preceded by read syncing leaders which force the READ sequence into sync very quickly. These leaders consist of a series of 111111110 or 1111111100 data streams. They are written by storing $FF in the data register in 36-cycle loops (111111110) or 40-cycle loops (1111111100) before flowing directly into the 32-cycle data writing loops. Writing data at intervals greater than 32 cycles results in trailing ZEROs (see WRITE sequence, State 2:39-SL0). This causes the ZEROs behind the eight ONEs in the read syncing leaders.

When a string of read pulses from a read syncing leader is applied to the sequencer, bytes of data following the seventh FF36 or fourth FF40 will always be in sync.* RWTS 3.2 syncing leaders are series of FF36s. RWTS 3.2 uses more than seven, which works fine, but only seven are necessary. RWTS 3.3 and DIIDD use FFs written in 40-cycle loops as read syncing leaders. A string of four FF40s followed by valid data will ensure that following data will be in sync. The use of FF40s allows RWTS 3.3 and DIIDD to sync in a shorter period of time, slightly increasing the space available for data.

There are two different times when data is written to a disk. One time is when RWTS, FILER, or other utility formats the disk, writing sector information for 16 (3.3/ProDOS) or 13 (3.2) sectors on 35 tracks. The other time is when the data field for a sector on a track is written. Writing a data field consists of positioning the head, then reading until the specified sector **address field** is found. After the desired address field has passed by, the **data field** is written from a 342-byte (410 if DOS 3.2) buffer.

The 3.2 and 3.3/ProDOS sector formats are shown in Figure 9.15. Other than the three extra sectors, there are several basic differences. The address field identifiers are different. D5 AA B5 in 3.2 and D5 AA 96 in 3.3/ProDOS, causing bootstrap incompatibility between the two formats. The read syncing leaders are different as was mentioned. DOS 3.2 overwrites each track with 9984 FF36s before writing the sectors, but DOS 3.3 and ProDOS don't. DOS 3.2 reserves space for data by writing 431 FF32s after each address field while formatting. DOS 3.3 and ProDOS reserve space by actually writing a data field, with unwritten gaps of about 50 MPU cycles on either side. The gaps are too short to enable accidental detection of a false data field identification string so they don't hurt anything. In either DOS 3.2 or DOS 3.3 processing, the data space behind an address field is overwritten with a leader and data field when a "write sector" call to RWTS is made. The same is true of a "write block" call to DIIDD, but the data space behind two address fields is overwritten.

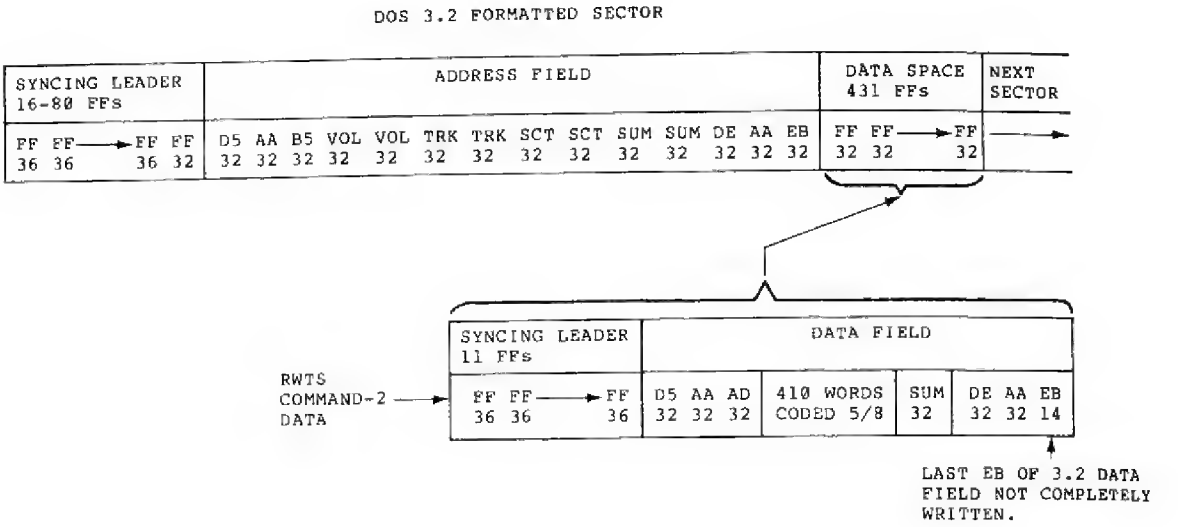
An interesting point about DOS formats is the DE AA EB series that follows every address field and data field. Apple has always had trouble writing the EB. In RWTS 3.2 they cut off the EB at the end of the data field by neglecting to wait 32 cycles before switching READ/WRITE to READ after storing EB in the data register. They changed that in RWTS 3.3, so the EB is actually written at the end of the **data field**. However, RWTS 3.3 and the FILER formatting routine both cut off the EB at the end of the **address field**. Those cut-off EBs are not really written, and no attempt to verify their presence is made in RWTS or DIIDD processing.

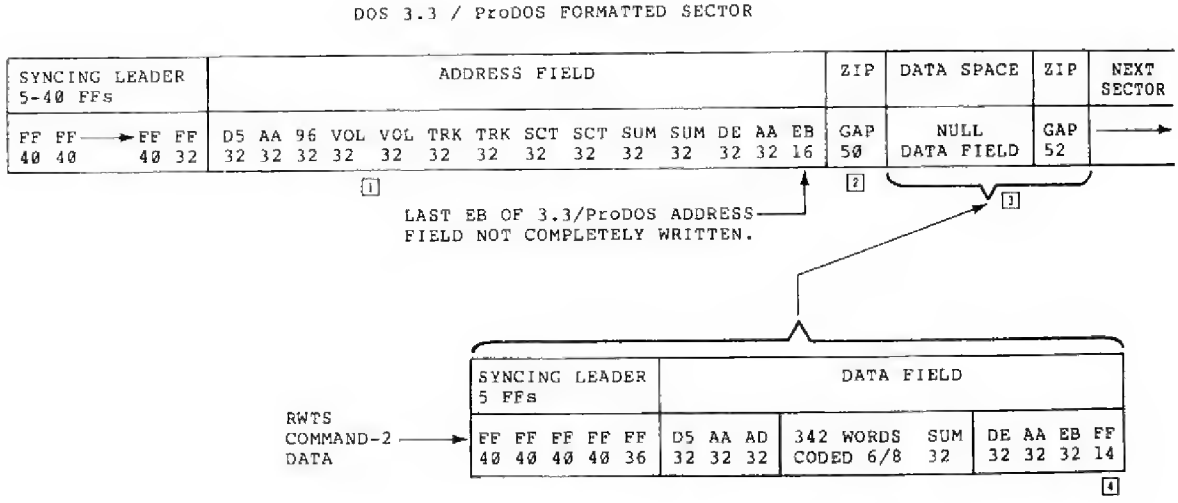
The READ Sequence

There is an odd contrast in Apple Disk II I/O. The 6502 program works the tail off the MPU to write data—initializing the WRITE sequence, then storing coded data in precise timing loops. Yet, the WRITE sequence listing is really pretty simple. Reading is just the opposite. The MPU program sits back and lets the sequencer do all the work. The program merely polls the data register, waiting for the sequencer to lay a complete byte at its feet. **The READ sequence is not simple.** Our discussion will concentrate on the 3.3 sequencer, which is very nearly the same as the 3.2 sequencer.

We start by assuming that the sequencer is configured for reading ($C08E,X; $C08C,X) and that a valid data field is passing across the READ/WRITE

*FF36 and FF40 refer to $FF bytes written using 36- or 40-cycle loops.





NOTES:
1. Volume = $01 ($AA $AB in 4-4 code) on disks formatted by FILER.
2. Pre-Data gap is 49 cycles in disks formatted by FILER.
   Post-Data gap is 54 cycles in disks formatted by FILER.
3. RWTS 3.3 Command 4/Command 2 data field misalignment is approximately 60 cycles.
   FILER format/DIIDD Command 2 data field misalignment is approximately 16 cycles
4. Final $FF is cut off at 14 cycles when data field is written by RWTS 3.3 Command 2 or 4 or by FILER formatting routine. It is cut off at 22 cycles when the data field is written by DIIDD Command 2.

Figure 9.15 Diskette Formatting.

head with every eighth bit set. Further assume that QA of the data register is not set, there is no read pulse present, and the sequencer is at State 2. You've got to start somewhere, and we are starting at State 2 of column 2 in the READ sequence listing of Figure 9.11.

At this location in the sequencer there is a 38-NOP. At State 3, there is a 48-NOP. At State 4, there is a 58-NOP. We are sequencing through the states, waiting for a read pulse. Assume a read pulse occurs at State 6, switching the sequencer to column 1. The read pulse will last for one sequencer clock because it is synchronized to the clock by a pair of flip-flops and a NAND gate. There is a D8-NOP at State 6 in column 1. In fact if you look at column 1, a read pulse at any of the states would have resulted in a D8-NOP.

When the read pulse goes away after the next clock, the sequencer goes to State D in column 2, a 08-NOP. This means move down to State 0 (18-NOP) and then up to State 1 (2D-SL1). This SHIFT LEFT ONE is a direct consequence of the read pulse. A read pulse occurred, so a ONE was shifted in. Assume the SL1 does not cause QA to become set, and don't get tired of assumptions. We now sequence to State 2 in column 2, right where we started, moving down the line, waiting for a read pulse.

This time let's say no read pulse occurs before we reach State 9. This is the point at which the sequencer decides it can't wait for a ONE any more—that was a ZERO bit. State 9 is a 29-SL0. A ZERO is shifted in. We'll say QA is still not set and we're back to State 2, waiting for a read pulse. This cycle will continue until QA becomes set after an SL0 or SL1. The sequencer is shifting in data based on the presence or absence of read pulses.

Now assume QA sets as the result of an SL0 or SL1. This breaks the loop, shifting flow to State 2 of column four, a 28-NOP. We are at State 2; the next state is State 2; we are going nowhere until a read pulse occurs. This is the **QA WAIT** location, outlined in both the 3.3 and 3.2 listings. **If the sequencer is in sync with the data stream, the fact that QA is set means that a valid 8-bit word is now in the data register** just as it was when it was stored there to be written. We will assume for now that the sequencer is in sync with the data stream. This means that the next read pulse will be the MSB set pulse of the next word.

So we're sitting at QA WAIT waiting for the BYTE FLAG of the next 8-bit group. The read pulse occurs. Do we clear the data register and do an SL1? No way, Jose. That's a valid byte sitting in the data register. We're going to hold that information as long as possible so that the 6502 program can figure out it's good stuff. The read pulse shifts the sequencer to State 2 of column 3 (08-NOP). Then the read pulse goes away and we sequence to column 4, State 0 (18-NOP), then State 1 (38-NOP), then State 3 (48-NOP), etc. We are sequencing now, waiting for the read pulse that means the second MSB is a ONE or the decision point that means the second MSB is a ZERO.

We'll say a read pulse occurs at State 8. The sequencer goes to column 3, State 8 (D8-NOP), column 4, State D (E8-NOP), State E (F8-NOP), State F (E0-CLR). **The data register is finally cleared.** It was held from the last read pulse or decision point of the previous word until past the BYTE FLAG pulse and second MSB pulse of this word. The sequencer then goes to column 2, State E (FD-SL1), then to column 2, State F (4D-SL1). **We shift ONE twice, once for the BYTE FLAG and once for the second MSB set**, then flow to State 4 of column 2 in the exact condition in which we started: QA reset, sequencing along, waiting for a read pulse.

Now we shift in six more bits of data which sets QA and puts us at QA WAIT. You should notice that QA is set precisely when a complete 8-bit word, lead by the BYTE FLAG, is completely shifted in. We made an assumption earlier that we were in sync, but no further assumptions were required. **Once we are in sync with a continuous stream of MSB set data, we stay in sync.**

We are at QA WAIT, waiting for the BYTE FLAG pulse that starts the next word. Suppose somebody had written the word we just read using a 36-cycle loop instead of 32. There would be a ZERO following the eight bits of data prior to the BYTE FLAG pulse. **You can't read ZEROs from QA WAIT.** There is no decision point here. The only thing the sequencer will respond to is a read pulse, so the ZERO passes right by and is not shifted to the data register.

Assume the next BYTE FLAG pulse occurs, this time followed by a ZERO. From QA WAIT the sequencer takes the same path it did previously, except no read pulse occurs. The decision point is reached at column 4, State C (A0-CLR) followed by column 2, State A (BD-SL1), State B (59-SL0), State 5 (68-NOP), etc. The sequencer cleared the data register, shifted a ONE, shifted a ZERO, then continued processing of the next six bits.

The whole idea of the QA WAIT is this: **the sequencer always knows the next bit is a ONE so it is not monitoring the next pulse as data; it is monitoring the next pulse as the BYTE FLAG.** It

monitors the pulse that follows the BYTE FLAG as data, and after monitoring this second pulse, it clears the data register and shifts in a ONE,ZERO or a ONE,ONE. In the process, the valid data word is held in the data register for a long time with the MSB set, a condition which a 6502 program can easily detect.

How does the sequencer get in sync with a data stream? The QA WAIT will cause the sequencer to eventually sync on nearly any valid data stream it encounters. This is because it ignores ZEROs while sitting at QA WAIT. **What the READ sequence does is to give the MPU a look at the data stream in groups of eight bits. Every such group has a leading ONE. ZEROs between a group and the next ONE are lost.**

Suppose the sequencer encounters a stream of data which was written in 32-cycle loops with the MSB set on every 8-bit word. Being out of sync, the sequencer groups the first data into eight bits lead by a ONE in some random way. This is illustrated in the first entry of Table 9.4, 1XXXBXXX. The B represents the BYTE FLAG pulse. It is a normal read pulse, like that generated by any other ONE, but it is represented by B here for illustration. When the sequencer is in sync, the BYTE FLAG will be the first ONE of every group.

At the first entry of Table 9.4, the BYTE FLAG is in the fifth bit position from the left. If the bit following this group is a ONE, that ONE becomes the first bit of the next group, and the BYTE FLAG stays in the fifth bit position. If, however, the bit following this group is a ZERO, the ZERO is lost and the BYTE FLAG moves closer to the MSB of the next group. Eventually, several ZEROs will have been encountered between groups, and the BYTE FLAG will reach the MSB. **From that point, the sequencer will stay in sync because the bit following each group of eight will always be a ONE.**

In data written by RWTS or DIIDD, the sequencer is never left to randomly sync on a data stream. All data is preceded by read syncing leaders which ensure that the sequencer is in sync when following data is encountered. A string of seven or more FFs written in 36-cycle loops or four or more FFs written in 40-cycle loops will ensure synchronization. These 9- and 10-bit write cycles cause synchronization because they are longer than the 8-bit groups. When encountered, these strings quickly are aligned into groups of eight ONEs followed by one or two ZEROs. Table 9.4 shows the worst case conditions for syncing to strings of FF36s and FF40s.

In the READ sequence examples we went through, many events could have occurred which were not taken into account. It would not be practical or useful to try to step through all possible events. Deeper analysis shows that the read sequence is designed to correctly interpret read pulses while tolerating an expected variation in the pulse interval. Figure 9.16 is my attempt to put the basic flow of the 3.3 READ sequence in perspective in a simple diagram. The pertinent sequencer states are listed next to each block to aid readers in correlating the flowchart to the sequencer listing.

In this flowchart, it was assumed that the read pulse interface circuits were successful in producing a read pulse which was actually one clock period in width. In the READ sequence, there are provisions to handle the rare event that a 2-clock read pulse occurs. This is an extra ounce of reliability which would clutter up the flowchart and obstruct understanding of normal flow. I have monitored the read pulse with an oscilloscope and have never seen any read pulses that were not properly synchronized to the sequencer clock.

Please direct your attention to the 8CP WAIT decision block near the bottom of Figure 9.16. This

Table 9.4 Syncing the Read Sequence to Data.

SYNCING TO A RANDOM DATA STREAM	SYNCING TO FF36s	SYNCING TO FF40s
1XXXBXXX	10B11111	100B1111
1XXXBXXX0	110B1111	11100B11
1XXBXXXX	1110B111	1111100B
1XXBXXXX	11110B11	111111100
1XXBXXXX00	111110B1	B111111100
1BXXXXXX	1111110B	B111111100
1BXXXXXX0	11111110	
BXXXXXXX	B11111110	
BXXXXXXX	B11111110	

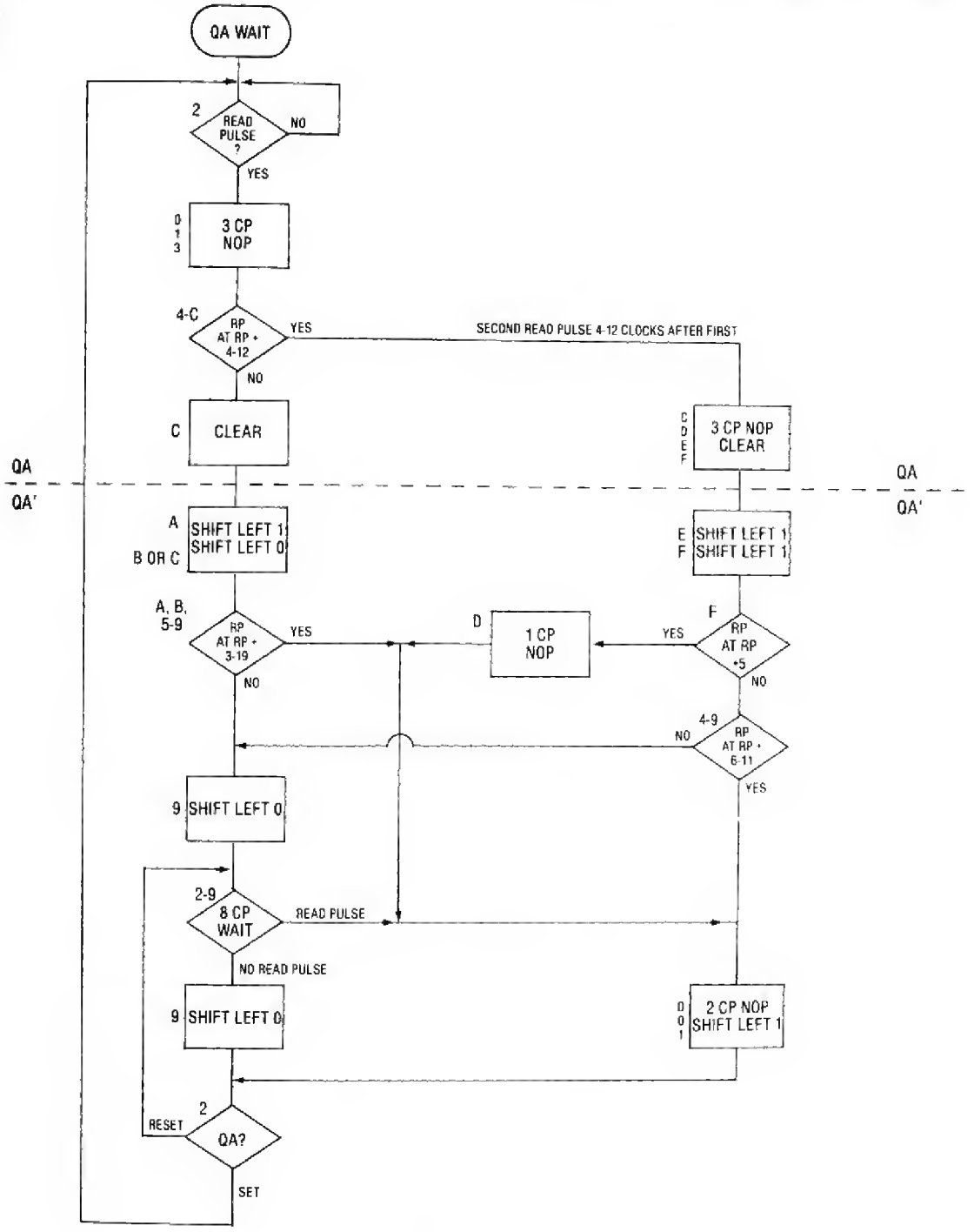


Figure 9.16 Simplified Flowchart of the Read Sequence.

represents the time when the sequencer is sequencing up column 2, waiting for a read pulse. The 8CP WAIT indicates that a ZERO will be shifted if a read pulse hasn't occurred by the eleventh sequencer clock after a read pulse, and following ZEROs will be shifted every eight clocks after the first ZERO. The write interval is eight clocks, of course, so there is an allowable distortion of three clock pulses for the first pulse position after a read pulse. This is the main difference between the 3.2 sequence and the 3.3 sequence. After the first ZERO, following ZEROs are shifted every 10 clocks in the 3.2 sequence. This represents a skew away from the write interval while reading strings of ZEROs. It makes no difference with the 3.2 data format, but it makes reading less reliable with the 3.3/ProDOS format.

The skew in the 3.2 READ sequence is shown in Figure 9.17. While the 3.3 sequencer always makes a shift 0 decision on the third clock period past an expected pulse, the 3.2 sequencer starts making the decision at the wrong point after the first ZERO. There are never more than two ZEROs in a row in 3.3/ProDOS data, but the sequencer will handle more than two if drive improvements make it a possibility. Note that the 3.2 sequencer will read 3.3/ProDOS formatted data, especially if the reading drive is slightly slow. The 3.3 sequencer reads 3.3/ProDOS data more reliably though.

If a read pulse occurs on the twelfth sequencer clock after the previous read pulse, it is smack in the middle of the two points where the sequencer expects a pulse. Is this an early pulse caused by a fast drive or a late pulse caused by a slow drive? The sequencer treats this pulse as an early pulse when it occurs, no doubt because it takes less room in the sequencer ROM to do so. As a result, the sequencer tolerates a fast **reading** drive or a slow **writing** drive better than the opposite condition when reading the data format it was designed to read.

For reference, I have tabulated the intervals which the sequencer can tolerate for various types of written data. Figure 9.18 shows this tabulation, and you can see that it shows what would happen with one data stream not used in any Apple DOS (B10001). This is sequencer performance, not drive and disk performance. The use of a string of three ZEROs is not recommended, although a copy protect scheme might use such a string.

As an example of interpreting Figure 9.18, interval A is the case of a BYTE FLAG pulse followed by a second MSB pulse. The expected interval between these two pulses is eight sequencer clocks. The 3.2 sequencer will read the second pulse correctly if it occurs anywhere from 3 to 12 clocks after the BYTE FLAG pulse. The 3.3 sequencer will read the second pulse correctly if it occurs anywhere from 4 to 12 clocks after the BYTE FLAG pulse.

The amount of time a valid data word is held in the data register depends on the second MSB of the next word, normal pulse interval variations, and, in the 3.2 sequencer, the least significant bits of the word. The only bad thing that can happen is that the data will be valid for too short a period of time due to a fast reading drive, a slow writing drive, or both. The average data valid period for some data streams is tabulated in Table 9.5. Values are in sequencer clocks, so the number of MPU cycles is half as many. In the Table 9.5 DATA STREAM entries, "B" represents the BYTE FLAG pulse which follows the valid data. As always, variations in disk surface speed at the read/write head are most likely to cause errors in the presence of one or two consecutive ZEROs.

The average data valid period must be at least 14 sequencer clocks if the MPU is to detect it in a normal 6502 polling loop. Notice that the 3.2 sequencer would have trouble meeting this requirement on data which has two trailing ZEROs. This is one more reason that a 3.2 sequencer would have an easier time with 3.3/ProDOS formatted data if the reading drive were a little slow.

The selection of D5 AA as the field identifier in DOS 3.2, DOS 3.3, and ProDOS was no accident. D5 and AA both consist of alternating strings of

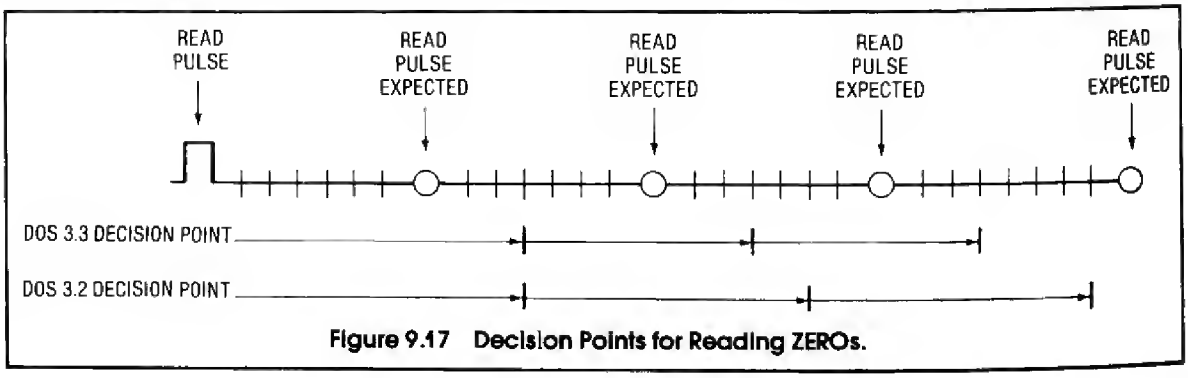
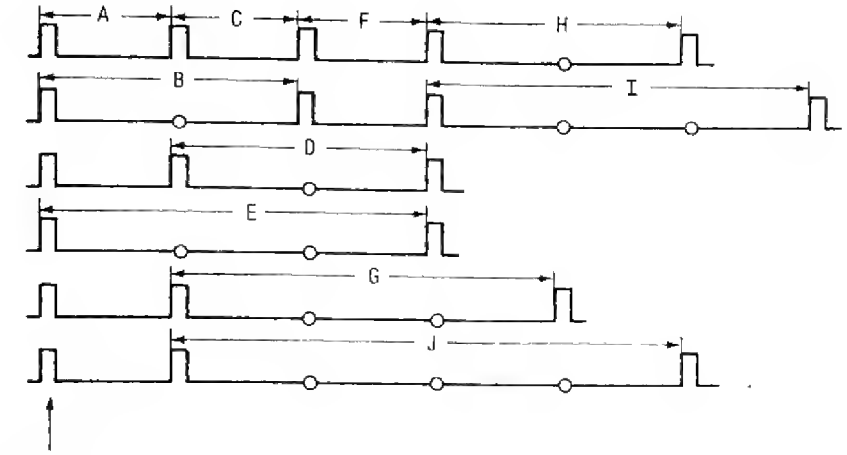


Figure 9.17 Decision Points for Reading ZEROs.



	DOS 3.2			DOS 3.3		
	MIN	AVG	MAX	MIN	AVG	MAX
A	3	8	12	4	8	12
B	13	16	21	13	16	19
C	5	8	11	5	8	11
D	12	16	21	12	16	19
E	22	24	31	20	24	27
F	2	8	11	4	8	11
G	22	24	31	20	24	27
H	12	16	21	12	16	19
I	22	24	31	20	24	27
J	32	32	41	28	32	35

ALL VALUES IN SEQUENCER CLOCK PERIODS

Figure 9.18 Read Performance of the Logic State Sequencer.

ONEs and ZEROs, a fact which gives them a unique identity in an environment filled with strings of FFs and other valid DOS data. Even when it is not in sync, the sequencer should never accidentally produce the D5 AA combination from a valid data stream. This is because the 1101010110101010, 11010101010101010, and 11010101001010101010 combinations do not exist in a valid data stream. Obviously, if D5 AA should not be accidentally read, then the D5 AA AD, D5 AA B5, and D5 AA 96 combinations should not be accidentally read either. Reliability is even greater because the AA AD combination itself should not be accidentally produced by DOS 3.2, DOS 3.3, or ProDOS data.

The AA B5 combination should not be accidentally produced by DOS 3.2 data, but it can be produced by an out-of-sync encounter with DOS 3.3 or ProDOS data. Specifically, the strings EA 96 AX or EA 96 BX can be grouped as AA B5 if the sequencer is out of sync:

```
X11.101010100.10110101.X
```

It is my speculation that this is the reason the address identifier was changed to D5 AA 96 in DOS 3.3, causing the DOS 3.3 controller to be unable to boot 3.2 disks. It is my further speculation that this is why DOS 3.3 data words all have at least one pair of adjacent ONEs in bits 0 through 6. This eliminates longer strings of ONEs alternating with ZEROs which might tend to be interpreted as field identifiers in an unstable read pulse environment. In particular, the data EA A5 9X or EA A5 AX or EA A5 BX could be grouped as AA 96 if the use of A5 were allowed:

```
X11.101010110.10010110.X
```

Table 9.5 Data Valid Periods in Sequencer Clocks.

DATA STREAM	3.2 AVERAGE VALID PERIOD	3.3 AVERAGE VALID PERIOD
XX1B1	18	16
X1ØB1	16	16
1ØØB1	14	16
XX1BØ	19	17
X1ØBØ	17	17
1ØØBØ	15	17

The switch from B5 to 96 as an address field identifier in DOS 3.3 may have been required to maintain the normal level of reliability in Apple Disk II I/O. I hope it was worth the cost of bootstrap incompatibility between DOS 3.2 and DOS 3.3. If so, my apologies to Apple for suggesting they could have done better to stick with D5 AA B5 as the address field identifier.

The Read Sequence as a Finite State Automaton

A reviewer of a rough draft of *Understanding the Apple II*, engineer/programmer Jim Aalto, used the Figure 9.11 listing of the DOS 3.3 read sequence to construct his own illustrative tool for studying the read sequence. This tool is valuable enough that we decided to include it in that book as well as this one (see Figure 9.19). Jim depicts the read sequence as a "finite state automaton." Like a flowchart, Figure 9.19 shows the logical paths the sequencer may take, but the flow is in step with the sequencer clocks pictured at the top. The average read pulse interval is also pictured, so sequencer performance with pulses arriving at various intervals is clearly illustrated. It is recommended that readers studying this figure attempt to relate it to the read sequence listing of Figure 9.11.

PROGRAMMING EXAMPLES FROM RWTS

There are several levels at which you can program disk I/O. The Apple disk operating systems are set up with very versatile file handling capabilities which can be utilized from BASIC as shown in the Apple manuals, *DOS Programmer's Manual for II, II+, IIe* and *BASIC Programming With ProDOS*. If one had a desire, he could also perform such direct control functions as turning drives on and off, selecting drive 1 or 2, positioning the head, checking for write protection, and checking to see if a drive is turned on from BASIC via PEEK instructions. As an example, the following Applesoft subroutine will tell you if a disk in Slot 6, Drive 1 is write protected:

```
10 SLOT6 = 49376 :  REM  $C0E0
20 DRIVE1 = PEEK (SLOT6 + 10)
30 REED = PEEK (SLOT6 + 14)
40 DRIVESTART = PEEK (SLOT6 + 9)
50 LODE = PEEK (SLOT6 + 13)
60 WPROTECT = PEEK (SLOT6 + 8)
70 REM  LINE 60 GETS DATA REGIS-
   TER AND TURNS DRIVE OFF
80 IF WPROTECT > 127 THEN PRINT
   "WRITE PROTECTED"
90 IF WPROTECT < 128 THEN PRINT
   "NOT WRITE PROTECTED"
100 RETURN
```

Note that no DOS need be resident for this program to work. It bypasses the DOS and goes straight to the controller.

More sophisticated programs may make direct use of DOS subroutines to perform special functions. You do not have to be an expert on DOS 3.3 or ProDOS to do this in your programs. *DOS Programmer's Manual for II, II+, IIe* shows you how to read a sector, write a sector, position the head to a track, or format a disk by making calls to RWTS. *ProDOS Technical Reference Manual (for the Apple II family)* shows you how to read and write ProDOS data blocks by making direct calls to DIIDD. It is also possible to make direct calls to the higher level file handlers, the DOS 3.2/3.3 file managers and the ProDOS MLI (Machine Language Interface). Information concerning the calling of these handlers is contained in the "beneath" books, *Beneath Apple DOS* and *Beneath Apple ProDOS* by Don Worth and Pieter Lechner.*

**Beneath Apple DOS*, Quality Software, 1982; *Beneath Apple ProDOS*, Quality Software, 1984. These books cover considerably more subjects than those mentioned here and are recommended reading for any student of Apple disk I/O. *ProDOS Technical Reference Manual* also contains information on calling the MLI and other programming tasks related to ProDOS and is recommended reading.

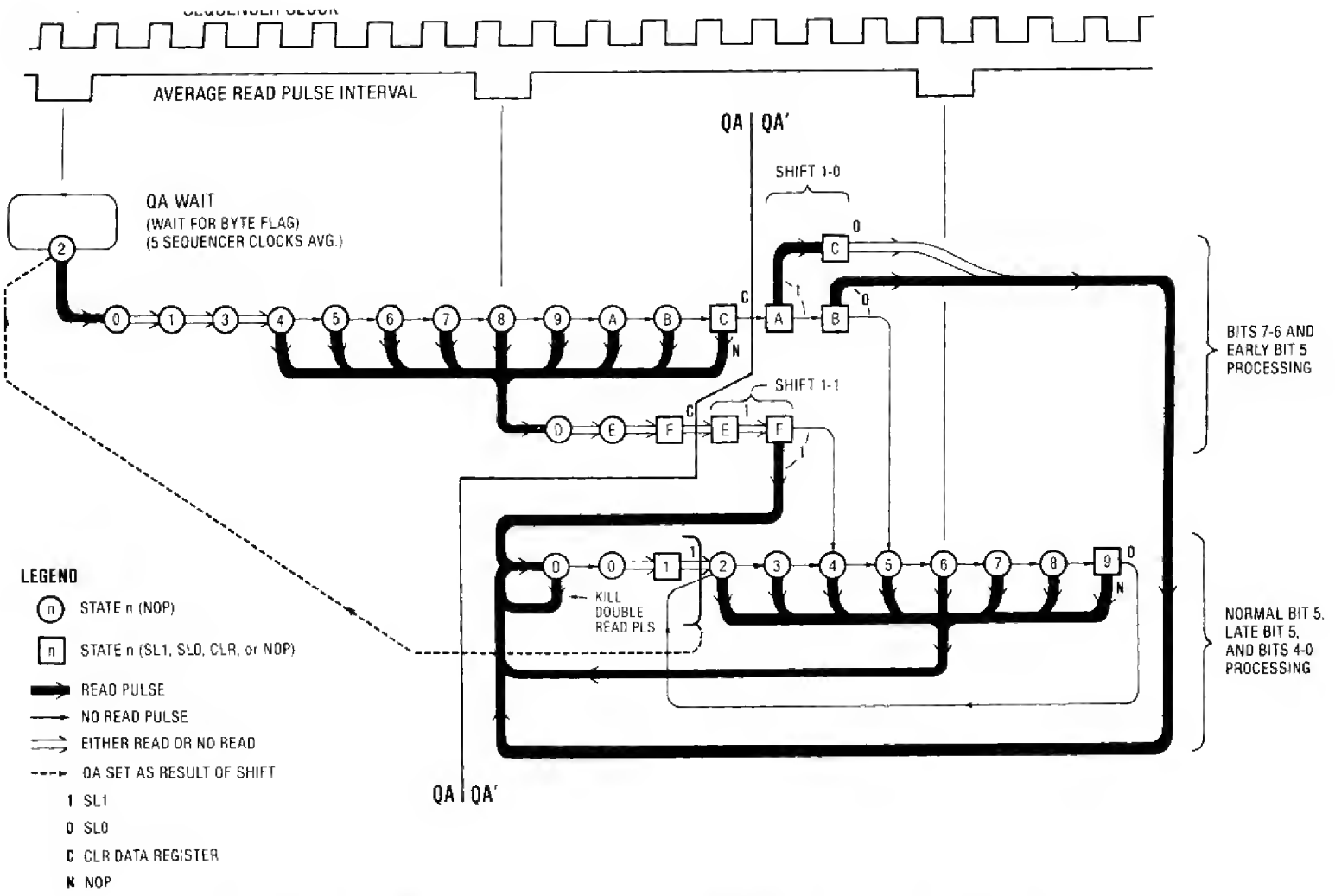


Figure 9.19 Jim Aalto's Finite State Automaton Diagram (DOS 3.3 Read Sequence).

On a different level, programs may bypass RWTS and DIIDD or modify them, substituting different routines to manipulate the controller and transfer data to and from the disk in formats not utilized by DOS 3.3 or ProDOS. To learn how to do this, you should study this chapter to see how the controller works, and you should study RWTS and DIIDD to see how DOS formats are written and read. RWTS and DIIDD are similar but different programs that illustrate some of the very complex techniques of reading and writing disk data. The following discussion points out some examples from RWTS 3.3 which show how disk data transfer can be accomplished. Some of the different features of DIIDD are discussed in the section following the RWTS descriptions.

RWTS is located at $B800—$BFFF of the DOS (assuming 48K of RAM). Its entry point is $BD00, although it can be called by a "JSR $3D9" in order to disable interrupts during RWTS processing and to ensure compatibility with future versions of DOS. Upon return from RWTS, 6502 status is interpreted as an error flag (carry set indicates that an error occcurred in RWTS). You are urged to make a listing of RWTS using your printer, so you can refer to it while reading this section. Make a listing of the Bootstrap ROM at $Cn00—$CnFF while you're at it. Reference to the RWTS general flowchart in Figure 9.20 should also help you keep your bearings.

Drive Turn-On

Drive turn-on is easy. Always configure for reading first (LDA $C08E,X), select the drive ($C08A,X/$C08B,X), and turn it on ($C089,X). Wait about a second after turn-on for the disk to get up to speed and then you can read and write. The RWTS turn-on procedure, which begins at $BD00, is a good deal more sophisticated than this. It takes into account all sorts of factors to get optimum performance in a general purpose routine.

First, RWTS checks to see if the slot being accessed is the same as the slot that was accessed last time RWTS was called ($BD13). If not ($BD19), it makes sure that a drive in the last slot accessed is not still rotating before proceeding. Remember that it takes one second for a drive to turn off after access to $C088,X. RWTS will not turn on two drives at once, presumably because of loading on the +12V power line.

It is possible to check whether a drive at a slot is on by configuring for reading data and monitoring the data register. If a drive is turned on, the data register will be changing and vice versa. This is the check used by RWTS:

```
            ORG   $BD22
            LDA   $C08E,X     READ
STILLON     LDY   #$08
            LDA   C08C,X      SHIFT
NOTSURE     CMP   C08C,X
            BNE   STILLON
            DEY
            BNE   NOTSURE
```

This routine loops until the drive at the previous slo turns off. It will hang in this loop until RESET i pressed if a call is made to RWTS that specifies a new slot and the last slot was never turned off RWTS itself always finishes by turning off the accessed drive.

After processing the old slot, RWTS checks if the new slot has a rotating drive ($BD34). This will be the case if the 1-second turn-off delay hasn't elapsed. If the drive is already rotating, there is no need to wait for it to get up to speed. RWTS saves the rotating/not rotating status ($BD4E), then turns the drive on ($BD4F). This prevents a still rotating drive from turning off after its 1-second lease on life.

Next, RWTS checks to see if the specified drive is the same as the last call to RWTS ($BD6A). If not ($BD6E), it then assumes that if a drive was rotating earlier, it was the wrong one. Therefore, it sets the rotating/not rotating status to not rotating ($BD73). It also selects the drive via the $C08A,X/$C08B,X switch ($BD74).

Now if the selected drive was not previously rotating, RWTS waits 150 to 175 milliseconds ($BD85) then calls the head positioning routine. 150 milliseconds is not enough time that the drive is up to speed, but RWTS saves time by positioning the head while waiting for drive speed to stabilize. The 150-millisecond delay accomplishes two things. First, it avoids trying to position the head just after a drive has been turned on, which is a period of heavy current flow on the +12 volt line. Second, if the opposite drive has just been disabled, the 12 volts may not have yet bled off from the disabled drive.* This might accidentally cause positioning of the disabled drive if RWTS tried to step the enabled drive too soon.

Before DOS 3.2.1, this delay before positioning did not exist in RWTS. Apple added it in DOS 3.2.1, presumably to improve performance, but they botched it up. The way it is written in DOS 3.2.1, the delay before head positioning is dependent primarily on the random state of $46E6 when RWTS is called and to a lesser extent on $46E6,X. The error is in the JSR $BA7F which is stored at $BD7E. This

*I measured +12V bleed off time in my Disk II drive at two milliseconds.

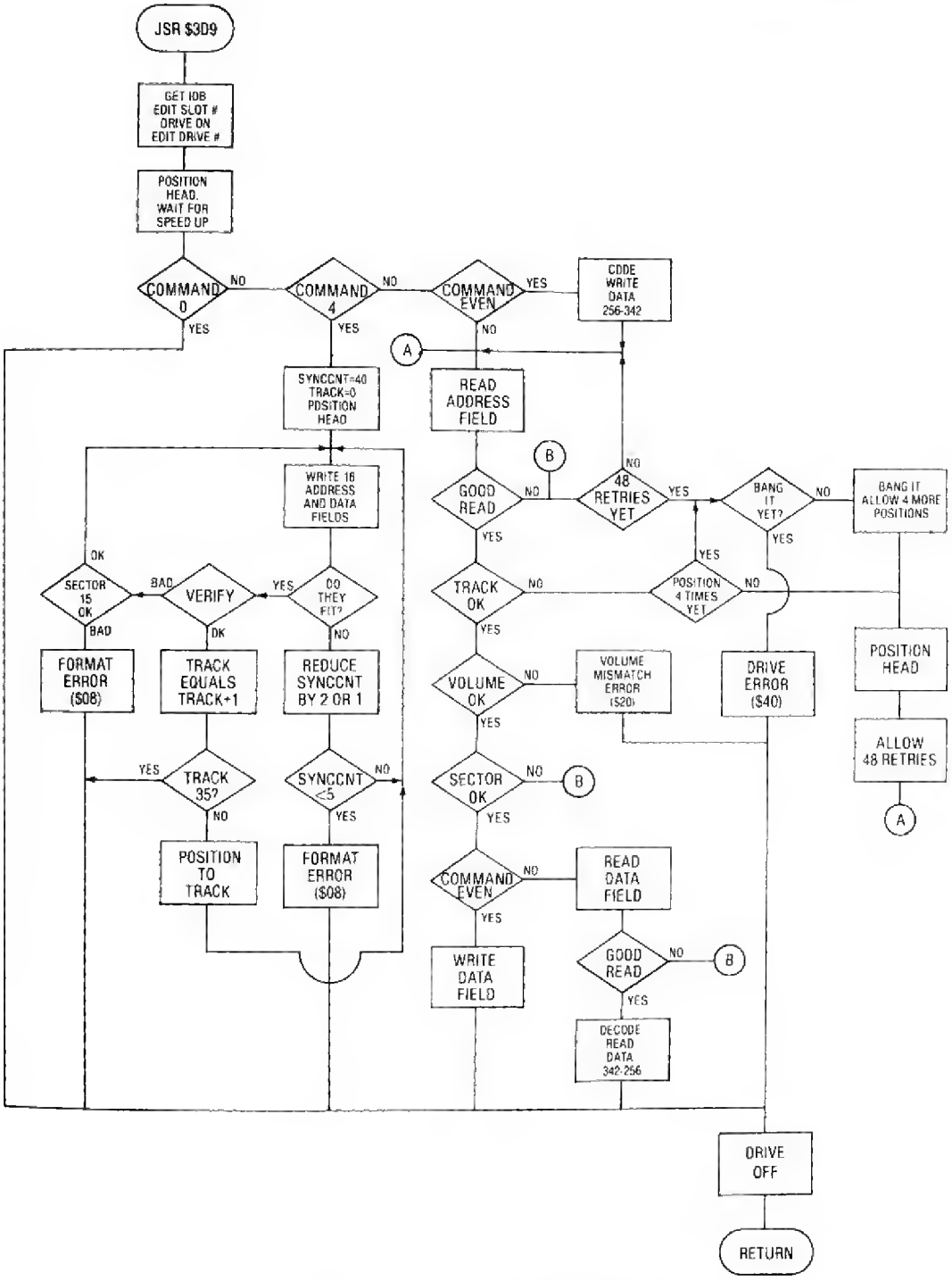


Figure 9.20 Flowchart of the RWTS Subroutine.

should be changed to JSR $BA7B to make the waiting subroutine entry point correct. The error does not exist in DOS 3.3 or ProDOS.

After the 150-millisecond delay in drives which were not previously rotating and almost immediately in previously rotating drives, the head is positioned to the selected track ($BD94). During the 150-millisecond delay and during head positioning, the motor-on time count is incremented at $46 and $47. This 2-byte counter counts the amount of time a drive has been rotating at the rate of one count per 100 microseconds. The preset count is part of the **Device Characteristics Table** used in a call to RWTS. The DOS uses a value of $D8EF which is equal to −$2711 or −10001 in decimal. **This converts to minus one second.**

After head positioning, the motor count will have partially counted up to $0000. If the drive was not previously rotating, RWTS will go no further than the count and wait loop at $BD9E until the motor count reaches $0000. This will complete the turn-on procedure, which takes one second plus small change for a not previously rotating drive and whatever time it takes to position the head for a previously rotating drive.

Positioning the Head in RWTS

At bootstrap time, the Apple finds track 0 by banging the head assembly against the outer stop. The programming sequence which does this is at steps $Cn3B through $Cn50 of the 3.3 Bootstrap ROM (the P5A ROM). This routine shows a very economical way to step the head in terms of software overhead. Just wait 20 milliseconds for motor response before turning off a stepper phase. The bootstrap routine uses 80 on-off descending references to $C0E0—$C0E7 (assumes Slot 6) to drive the head 38 or 40 tracks outward depending on initial phase alignment. The timing is $C0E0, $C0E1, wait, $C0E0, $C0E7, wait, $C0E6, $C0E5, wait, $C0E4, $C0E3, wait,. . ., $C0E4, $C0E3, wait, $C0E2, $C0E1, wait. The wait period is 20 milliseconds. Note that phase-0 is left energized on the stepper motor after positioning. This is indicative of the fact that even numbered tracks are phase-0 aligned and odd numbered tracks are phase-2 aligned. It also contradicts every theory I can think of as to why the analog card was designed so that leaving phase-1 on forces write protection.

The head does not have to be banged against the stop to locate its position. The track number is written as part of the address field in front of every sector on a formatted disk. The head location can be determined at any time by simply reading an address field. Of course banging the head against the stop is the best way to absolutely determine head position, and there is no room in the bootstrap ROM for a routine that reads an address field and then tiptoes out to track 0.

The RWTS positioning routine is far more sophisticated, and there are two calls you can make. Both calls are made with slot number times $10 in the X-register. You can do a "JSR $B9A0" with the destination track times two in the accumulator and the current track times two at $478. This will simply position the head using two phases per track. You can also do a "JSR $BE5A" with the destination track in the accumulator, a Device Characteristics Table set up, and some RAM locations correctly set up. This will edit the RAM locations and do a JSR $B9A0. The RAM assignments are:

$3C, $3D	—Device Characteristics Table location.
$35	—MSB set if drive 1, MSB reset if drive 2.
$478 plus slot #	—Drive 1 last accessed track times 2.
$4F8 plus slot #	—Drive 2 last accessed track times 2.

The $B9A0 routine is the actual positioning routine for either type of call. It uses a technique of programming duration periods of the stepper motor controls to maximize acceleration in the first part of head travel then to reduce head velocity near the destination track to prevent overshoot and minimize settling time. For this purpose, the routine utilizes a **wait after phase-on table** at $BA11 and a **wait after phase-off table** at $BA1D. These amount to momentum tables for a typical head assembly. The values in the table can be multiplied by .1 milliseconds to give the wait time.

As an example, the Slot 6 phase control for stepping from track $10 to track $11 is as follows: $C0E3, wait $01, $C0E0, wait $70, $C0E5, wait $30, $C0E2, wait $2C, wait $100, $C0E4. This is phase-1 on, phase-0 off, phase-2 on, phase-1 off, phase-2 off. The above wait periods in decimal add up to .1 + 11.2 + 4.8 + 4.4 + 25.6 = 46.1 milliseconds which is the single track response time of the Disk II operating with RWTS, not including a millisecond or so of general computing time. The final wait of 25.6 milliseconds doesn't come from the wait tables but from looping through the 100-microsecond wait routine ($BA00) 256 times at the end of every head positioning sequence. This is the settling time of the Disk II head positioning assembly.

As mentioned previously, the 100-microsecond waiting loop that is used to generate delay periods also increments the motor-on counter. This is part of the scheme by which the head is positioned while the motor gets up to speed, killing two welfare bills with one Republican.

Formatting the Disk (Command 4)

Once the head is positioned and the disk is up to speed, RWTS looks at the command entry of the **IOB** (I/O Block) to see what it is supposed to do ($BDAB). Command 0 ($BDAF) causes an immediate exit with drives off and no error indicated. Command 4 ($BDB3) causes the disk to be formatted with 16 sectors written on every track. Other than Commands 0 and 4, even commands cause **writing** of a sector's data field, and odd Commands cause **reading** of a sector's data field, but only Commands 2 and 1 are normally used. Command 2 and 1 processing is the part that gives RWTS its name.

The FORMAT routine starts at $BEAF. It works by starting at track 0 ($BEBB), then formatting each track one by one. It starts by guessing there will be 40 FFs in the read syncing leaders which precede every sector ($BED0). It then writes the 16 sectors with 128 FF40s before sector 0 ($BF0D) and 40 FF40s before the other sectors. Sectors are written in order from 0 to F, but they are effectively interleaved because the sector specified in the IOB is not actually the one that is read during a Command 1 or 2 call to RWTS. Rather, **the specified sector indexes the Sector Interleave Table at $BFB8.**

Writing a sector while formatting consists of writing the address field ($BF17) which is preceded by a read syncing leader, then writing a data field ($BF1C), which is also preceded by a read syncing leader. The write coded data buffer contains all ZEROs ($BEBB), which means the data in the data field will be a string of $96s.

After writing the last sector on a track, the MPU waits for a number of cycles equal to about 200 plus 50 times the number of sync bytes (50-cycle loop at $BF3A). This delay ensures that there will be a read syncing leader preceding sector 0 that is at least as long as those preceding the other sectors. An attempt is then made to read the address field of sector 0. At 40 sync bytes, the sector 0 address field will probably be long gone, in which case the size of the address field syncing leader will be reduced by two ($BF52), then the tail end of sector F will be found ($BF71), and the sectors will be written again starting at the same point on the disk as before ($BF0D). This cycle continues until the sectors fit evenly on the disk. The sync count is reduced by twos until it reaches 16, then by ones until it reaches 5. If 16 sectors do not fit on the disk with a 5-byte leader, the disk speed is probably adjusted way too high and a formatting error (error code $08) is signaled ($BF60).

When the sectors fit well on the track, all the address fields ($BF62) and data fields ($BF67) are read and validated. As each sector is validated ($BF6A), an FF is stored in the correct spot in the **Sector Initialization Map** at $BFA8. Examining this map may give you hints about the cause of formatting errors. When all the sectors are validated, the track number is checked ($BF98). If it is track 0 and the sync count is greater than 15, then two is subtracted from the sync count ($BFA2). Otherwise, the sync count is left alone for the next track. Since the optimum sync count is found while formatting track 0, the other tracks take much less time to format.

After a track is completely formatted, flow returns to $BEDC. Address fields are read ($BEEB) until sector 0 is found, then the sector 0 data field is read ($BEF4) and the head is stepped to the next track (or RWTS is exited with the drive off if all tracks have been formatted). The drive will actually turn off about one second after RWTS is exited.

Waiting until the sector 0 data field has just past before switching tracks means that, after reading the data field of sector F on a track, data can be processed during the following sector 0 period, then the head can be stepped one track inward and a sector 0 address field will be ready to read. This could be the basis for a very fast special purpose loader that reads data from disk while increasing sector number and stepping inward. It does not, however, particularly minimize DOS 3.3 access time because neither DOS 3.3 file access (decrease sector number while stepping inward or outward) nor booting (decrease sector number while stepping inward, then outward) takes advantage of it.

Reading and Writing Sectors (Commands 1 and 2)

Reading and writing sectors are very similar operations in RWTS. Both operations cause drive selection and turn-on, head positioning, location of pertinent sector, reading or writing of a data field, and drive turn-off. Additionally, write data is coded from 256 bytes to 342 6-bit words before locating the specified sector, and read data is decoded from 342 6-bit words to 256 bytes after reading a data field.

If the RWTS command is not a 0 or 4, read/write processing begins at $BDB5. First, the command

type is checked and saved ($BDB5). If a data field is to be written, the 256 bytes of data specified by the IOB (I/O Block) are coded into 342 6-bit words ($BDB9). After write data coding ($BDBC), read and write processing take the same path. A retry count is set to $30 indicating 48 attempts will be made to read the correct address field. The address field is read ($BDC4), then checked for correct track ($BDED), volume ($BE10), and sector ($BE26).

Unless volume 0 is specified, finding the wrong volume causes a return from RWTS with the drive off and a VOLUME MISMATCH ERROR indication (error code $20). Finding an incorrect track number causes up to four repositioning attempts (count preset at $BD09), followed by a major track recalculation ($BDCE), and up to four more repositioning attempts ($BDDC) before a DRIVE ERROR is indicated. A major track recalculation consists of banging the head against the track 0 stop, then repositioning to the specified track. Only one major track recalculation is allowed because the number of recalculation tries is set to one at the beginning of RWTS ($BD04). A major track recalculation is also performed if the correct sector cannot be located after reading 48 address fields. Another 48 attempts are made after recalculation before a DRIVE ERROR is indicated (error code $40).

Once the correct address field has been read, the command is checked again ($BE32). Read operations consist of reading the data field ($BE35), decoding the buffer 342 to 256 ($BE40), turning the motor off ($BE4D), and exiting. Write operations consist of writing the coded data to the data field ($BE51), turning the motor off ($BE4D), and exiting. It takes longer to begin writing than it does to begin reading, so the reading will start just before the read syncing leader is encountered if the data field was written by a Command 2 on the same drive.

Data fields written by Command 2 are not aligned with those written during formatting. Writing of the data field during formatting begins 50 cycles after writing of the address field ends. Writing of the data field during Command 2 begins 123 (118 if volume = 0) cycles after the DE AA is detected at the end of the address field. This is the equivalent point in time at which the 16-cycle EB is stored while writing the address field plus 0 to 6 cycles for MPU detection of AA. As a result, Command 2 begins writing a data field about 60 cycles (123 + 3 − 16 − 50) after the NULL data field is written while formatting.

There are only 52 cycles between the end of the NULL data field and the beginning of the syncing leader of the next address field, so the data field written by Command 2 will bump up against the syncing leader of the following address field. It seems likely that the first address field sync byte will be overwritten by Command 2. Furthermore, if the Command 2 drive is faster than the formatting drive, destruction of the first part of the address field read syncing leader is a certainty. This should cause no problem unless the formatting drive was very fast, causing very short address field leaders.

Command 1 should still be able to read the NULL data field written by Command 4. Command 1 will cause the MPU to start looking at the data register while the data field syncing leader is still passing the read/write head. The data field leader is 196 cycles long so the 60-cycle misalignment should not cause the data field identifier to be missed.

The misalignment between the Command 2 and Command 4 data fields is caused by the long processing time used in verifying volume, track, and sector numbers during Commands 2 and 1. If they were concerned, Apple could easily and substantially reduce the misalignment by fetching volume and sector from the IOB and Sector Interleave Table before reading the address field instead of after.*

The error detection circuitry in RWTS is very sophisticated, allowing as it does for the possible problems that might occur in data transfer. Not so sophisticated is the error indication found in the IOB after a return from RWTS. There are three types of error codes: VOLUME MISMATCH ($20), error during Command 2 or 1 ($40), and error during Command 4 ($08). With a little extra programming RWTS could give such indications as ADDRESS FIELD CHECKSUM ERROR, DATA FIELD CHECKSUM ERROR, CAN'T FIND ADDRESS FIELD IDENTIFIER, CAN'T FIND DATA FIELD INDENTIFIER, CAN'T FIND END OF ADDRESS FIELD, CAN'T FIND END OF DATA FIELD, CAN'T FIND TRACK, CAN'T FIND SECTOR, SYNC COUNT < 5, and so on. As it is, such DOS indications as DRIVE ERROR or I/O ERROR mean only that something went wrong in RWTS.

*In effect, Apple does this in DIIDD. The sector lookup is performed in the course of getting sector numbers from the block number, and there is no volume check in ProDOS. As a result, the misalignment between formatted data fields and data fields written by DIIDD is reduced from about 60 cycles in DOS 3.3 to about 16 cycles in ProDOS (78 + 3 − 16 − 49).

Write Routines

There are several routines related to writing in RWTS. One is the **WRITE ADDRESS FIELD** routine at $BC56. This routine writes the syncing leader and address field shown in Figure 9.15, and it is only called when a disk is being formatted. The input parameters are:

Y Reg—Number of FFs in syncing leader
$41—Volume
$44—Track
$3F—Sector
$3E—Contains the value $AA

The routine first checks for write protection ($BC57), then stores the first FF in the data register ($BC61), then continues to write FFs in a 40-cycle loop ($BC69—$BC77). The number of FFs is adjusted by the format routine so the 16 sectors fit on each track without a large gap between sector 0 and sector 15. The minimum number of FFs in the leader will be 5.

After the sync writing loop is exited, the series D5 AA 96 is stored directly to the data register at 32-cycle intervals. This is the address field identifier—the values D5 and AA are not used in the storing of data. The D5 is placed in the data register 32 cycles after the last FF of the syncing leader, so there are no ZEROs following the last FF and it serves no read syncing purpose.

The volume ($BC88), track ($BC8D), and sector ($BC92) numbers are written next, followed by a checksum which is the exclusive-OR of the volume, track, and sector numbers. These four values cannot be stored directly to the disk, but RWTS writes them in a pair of 32-cycle loops following a simple coding scheme. First, the value to be stored is shifted left and ORed with AA. After storing this result to the disk in a 32-cycle loop, the unshifted value is ORed with AA and stored to the disk. The result is that four of the bits are encoded in each storage cycle, and only valid data words are stored. There are 16 possible storage words in this **4-4 encoded storage format**: AA, AB, AE, AF, BA, BB, BE, BF, EA, EB, EE, EF, FA, FB, FE, and FF. The use of AA here slightly degrades the integrity of the D5 AA field identifier, but the system works anyway.

The 4-4 CODE AND WRITE routine begins at $BCC4. This coding method offers less density than the 6-8 coding method, but it could be the basis for a low overhead read/write subroutine which would transfer 2500-byte blocks of data directly between RAM and a track on the disk. Such a low overhead subroutine would serve the purposes of many Apple users.

After the checksum is written, the WRITE ADDRESS routine finishes up by writing the values DE and AA, then part of an EB. The EB is truncated to a 1110 since the controller's READ/WRITE switch is switched to READ ($BCBD) on the 16th MPU clock after the EB is stored in the data register. Switching to READ here results in a 50-cycle gap between the address field and data field. The 50-cycle gap causes no harm, because it is not long enough to randomly produce a 3-word data field identifier (D5 AA 96 or D5 AA AD).

Another write related routine is a routine which codes a 256-byte data block into 342 6-bit words. This **PRENIBBLIZE** routine begins at $B800. The address of the 256-byte data block must be stored at $3E and $3F and the 6-bit words will be stored in a pair of coded buffers in the 00XXXXXX format. The six MSBs of the 256 data bytes are stored in a 256-word buffer beginning at $BB00, and the 2 LSBs of the 256 data bytes are grouped together in an 86-word buffer beginning at $BC00. The 256-word and 86-word buffers are the source file for the write data field routine at $B82A.

The **WRITE DATA FIELD** routine is called in formatting a disk (RWTS Command 4) and in writing data to a sector (RWTS Command 2). In formatting, the 256-word and 86-word coded data buffers contain all ZEROs, so a NULL data field is written 50 cycles after the end of an address field. In a Command 2 write, data is coded using the PRENIBBLIZE routine first, then the desired address field is read, then the data is written with the WRITE DATA FIELD routine using the coded data buffers as a source file. This "real" data field is not centered on the NULL data field but lags it by approximately 60 cycles.

The WRITE DATA FIELD routine checks for write protection ($B830), writes four FF40s followed by an FF36 ($B83D), then stores the data field identifier, D5 AA AD, directly to the data register at 32-cycle intervals. Then the coded data buffers are written in 32-cycle loops, using the exclusive-OR of the current 6-bit word and the previous 6-bit word to index the Write Table at $BA29 to obtain the value to be written. This odd storage method is reversed in the read operation, and it creates a checksum by which the validity of data transfer is checked. The 86-word buffer is read first for output

from the top down ($B862), then the 256-word buffer is read from the bottom up ($BA7B). Afterwards, the DE AA EB trailer is written with the EB completed. This is opposed to the incomplete EB at the end of the 3.3/ProDOS address field and the 3.2 data field.

Read Routines

The **READ ADDRESS FIELD** routine is at $B944. This routine is used to locate any address field, fetch the volume, track, and sector, and to check validity of the read. It is performed in formatting to verify correct sector distribution and content, and it is used in reading or writing the data field of a sector to locate the correct sector.

The routine starts by looking for any D5 AA 96 sequence. This should occur within roughly 385 valid data register words from any point on the disk. If it doesn't occur within 772 valid data words ($10000 minus $FCFC), the routine is exited ($B94D) with the carry status set, indicating an error condition. After finding D5 AA 96, the four parameters are read in a 4-4 read loop while accumulating the checksum ($B96D). Volume, Track, Sector, and Checksum are stored at $2C, $2B, $2A, and $29 respectively. Next, the presence of the trailing DE AA is verified. Checksum failure or absence of DE AA causes the carry to be set, indicating an error condition. The calling routine will not process the data if the error flag is set or the volume, track and sector are not those desired. RWTS calling routines will attempt to find the correct sector 48 times, bang the head against the stop, then reposition, then try to find the correct sector 48 more times before giving up and deciding there is an error. Reading an incorrect volume, however, causes the immediate return with a VOLUME MISMATCH error unless volume 0 was specified in the IOB.

The sector which is read is not taken directly from the IOB. Rather, the IOB value is used to index the Sector Interleave Table at $BFB8 which contains 0, D, B, 9, 7, 5, 3, 1, E, C, A, 8, 6, 4, 2, F. As an example, if the IOB specifies sector 1, the sector which will be sought will be sector D. This leads to the following effective order of sectors on each track: 0, 7, E, 6, D, 5, C, 4, B, 3, A, 2, 9, 1, 8, F. Presumably it is chosen to minimize access time to sequential sectors in the DOS environment.

The **READ DATA FIELD** routine is at $B8DC. This is called when the sector writing is verified while formatting or when a Read Sector (Command 1) call is made to RWTS. Reading begins after the desired sector is located via the READ ADDRESS FIELD routine.

Since the READ DATA FIELD routine is always called after the address field has been read and verified, the data field should pass under the read/write head very soon. If more than 32 valid words are read ($B8DC) and D5 AA AD isn't found, the routine is exited with carry set to indicate an error. Oddly, finding D5 AA XX other than D5 AA AD gives the routine 86 extra chances to find D5 AA AD.

After finding the identifier, the data field is read into the 86-word buffer, top byte first ($B8FF) and into the 256-word buffer, bottom byte first ($B913). Each valid word read from the data register is used to index the Read Table which begins at $BA96. This table is the inverse of the Write Table. Each table value is exclusively ORed with the "running total" to get the value stored in the big and little buffers. This is the inversion of the writing process and the running total is checked for correctness at $B92A. A nonmatching checksum or absence of a trailing DE AA causes return with carry set, indicating a read error.

The formatting routine calls the READ DATA FIELD routine just to check the carry status and to verify its own handiwork. When reading the data field in a Command 1 call to RWTS, the data must be decoded from the 6-bit words in the big and little buffers into the 256-byte RAM buffer that was specified by the IOB. A **POSTNIBBLIZE** routine which performs this is located at $B8C2.

DIFFERENCES BETWEEN RWTS AND DIIDD

DIIDD performs the same sort of tasks in ProDOS that RWTS performs in DOS 3.3 and DOS 3.2. As you might expect, close study of these two programs reveals that DIIDD is a modified version of RWTS 3.3. There are, however, some basic differences in overall concept between DIIDD and RWTS as well as differences in the implementation of the read and write sector functions which make DIIDD read/write processing faster than RWTS read/write processing.

The difference in overall concept is in the types of calls that can be made to RWTS and DIIDD. RWTS supports head positioning (Command 0), sector reading (Command 1), sector writing (Command 2), and formatting (Command 4). DIIDD supports checking for write protection (Command 0), block reading (Command 1), and block writing (Command 2). Formatting must be performed via a separate utility (normally FILER) in ProDOS, and

reading or writing of single sectors is not normally performed.

There are some minor differences between the RWTS Command 4 routines and the FILER **format** routines ($7900—$7D37).* Most importantly, the tolerance on the size of the read syncing leader is tighter in the FILER routines ($10 to $1A as specified in memory locations $7D1F and $7D20 of FILER). This means that drive speed must be very close to 300 rpm if FILER is to format disks. An associated bonus is that the FILER formatting routine returns a fast (error code $34) or slow (error code $33) indication if a drive is off speed.

Calling DIIDD from a machine language program is considerably simpler than calling RWTS. The unwieldy, relocatable I/O block and device characteristics table of RWTS are replaced by a simple set of device driver input parameters at memory locations $42—$47:

LOCATION	DESCRIPTION	OPTIONS
$42	Command code	$00 = STATUS $01 = READ $02 = WRITE
$43	Unit Number	DSSS0000 D = Drive number (0 = drive 1, 1 = drive 2); SSS = Slot number (0 to 7)
$44-45	I/O Buffer	Can be $0000 to $FFFF
$46-47	Block Number	Must be $00 to $117

Except for these basic parameters, none of the features which are specified in a RWTS IOB and DCT can be specified for DIIDD. This isn't much of a disadvantage, though, because few persons require changes to standard DIIDD characteristics and those that do can always modify DIIDD code for their purposes.

Sectors vs. Blocks

The general RWTS scheme of coincident flow for reading and writing sectors is intact in DIIDD. Read and write flow is the same (except for write prenibblization) up to the time the specified track/sector address field is located. Different is the fact that a **Command 1 call to DIIDD results in the reading of two sectors to a 512-byte memory buffer**. Similarly, a **Command 2 call results in the writing of two sectors from a 512-byte memory buffer**. This is accomplished by code at the front end of DIIDD ($F800—F835) that converts the input **block number** to a track and sector number. The data field of that sector is then read to or written from the first half of the specified 512-byte buffer ($F823), and then the data field of that sector plus two is read to or written from the second half of the specified buffer ($F829).

Conversion of the block number to track and sector numbers consists of converting the 2-byte block word

0,0,0,0,0,0,0,T5 T4,T3,T2,T1,T0,S2,S1,S0

to the track and sector words

0,0,T5,T4,T3,T2,T1,T0 0,0,0,0,S1,S0,0,S2.

For example, the block words, 00000000 01101101, convert to track 00001101 ($D), sector 00000101 ($5), and a block $06D call to DIIDD will result in reading or writing the data block of track $D, sectors $5 and $7.

The nature of the sector transformation described above is such that it optimizes the aecess to blocks in the ProDOS environment. More specifically, the two sectors accessed by a call to DIIDD are separated by a single sector, and sequential blocks are separated by a single sector (except for seetor $C/$E blocks which are separated from the following blocks by two sectors). Therefore, ProDOS has one sector period (minus prenibblization time if writing) in which to process between calls to DIIDD if sequential blocks are to be accesssed without waiting for an entire disk revolution. This is also true when the sequential blocks are on two different tracks since track-to-track sector synchronization during formatting places one sector period (sector 0) plus a single track positioning period between sector $F on one track and seetor $0 on the next higher track.

After the track and sector are derived from the block number, the next steps are **drive turn-on** and **head positioning**. In this area, DIIDD is very nearly the same as RWTS except for one big operational change. The change is that DIIDD only waits about .6 seconds for drive speed stabilization compared to the 1 second waiting period of RWTS. This delay is determined by the fourth byte of the device characteristics table of RWTS ($D8 as specified by Apple literature) or by the load immediate instruction at $F84F of DIIDD (LDA #$E8). Roughly, the $D8 and $E8 values lead to ($100 − $D8) x .025 = 1 second delay plus change for RWTS and ($100 − $E8) x .025 = .6 second delay plus change for DIIDD.

I hopefully assume that, while developing ProDOS, Apple decided the old Shugart 1-second specification—hey Rockeeeey, watch me pull a rabbit out

*Program references in this section are to FILER and DIIDD listings from the ProDOS version 1.0.1 master diskette, Jan. 1, 1984. These addresses may change in future versions of ProDOS. Subtract $100 from FILER addresses given here for ProDOS versions predating version 1.0.1.

of my hat—was too conservative. I more hopefully assume that Apple tested a variety of disk drives, and settled on .6 seconds as a safe motor-on to read/write delay period. Of course, a typist might have just entered $E8 instead of $D8 by mistake, and thereby established a new Apple disk I/O specification. In any case, it will work for DOS 3.3 if it works for ProDOS, and you can speed up DOS 3.3 access by using $E8 instead of $D8 in the device characteristics table.

The head positioning routines of DIIDD are very close to those of RWTS, right up to the same values in the wait after stepper phase on and off tables. A very subtle difference is that the same Accumulator x .1 millisecond timing routine is used for producing stepping delays ($FB85), but that non-zero page memory locations are used for timing. The INC $FB6F and INC $FB70 instructions of the DIIDD subroutine take one cycle longer than the INC $46 and INC $47 instructions of the equivalent RWTS subroutine, resulting in a 102-cycle timing loop in DIIDD compared to the 101-cycle loop in RWTS. The net result is that head positioning with DIIDD takes about 1% longer than head positioning with RWTS. The slow head positioning does not noticeable decrease ProDOS disk access speed, but it does reduce allowable processing time between access to a sector $1/$F block on one track and a sector $0/$E block on the next higher track.

As noted before, DIIDD **read/write processing** is very similar to RWTS read/write processing. Notable differences are that after address field reading there is no volume check, the sector check is simplified ($F8D7), and reading an incorrect track number results in an attempt to step to the desired track from the located track ($F8C8). Also, after reading a data field, there is no call to a "postnibblization" subroutine.

The simplified sector check and absence of a volume check after address field reading result in reduction of the delay between Command 2 address field reading and data field writing from about 123 cycles in RWTS to about 78 cycles in DIIDD. This, in turn, reduces the misalignment between data fields written during formatting and data fields written via Command 2 from about 60 cycles in RWTS to about 16 cycles in DIIDD. Since there is a 54-cycle gap trailing the data field and the speed tolerance of ProDOS is so tight, the DIIDD Command 2 data field should not partially overwrite the read syncing leader of the following address field.

Nibblization Improvements

ProDOS disk I/O is much faster than DOS 3.3 disk I/O. Part of the reason for this overall speed improvement is a reduction of data processing time in Command 1 and 2 calls to DIIDD.* The main improvement is in read processing where data that is read is not stored directly to 256- and 86-word buffers for later "postnibblizing" as it is in RWTS. Instead, data is postnibblized "on the fly." That is, the 6-2 coded data is decoded immediately after it is read and is then stored directly in the 256-byte memory area indicated by the DIIDD input parameters. Reading data on the fly cuts DIIDD read processing time approximately in half (once the drive is up to speed). Additionally, write prenibblizing takes less time with DIIDD than RWTS, so there is some speedup in the write process as well.

RWTS read/write processing utilizes separate 256- and 86-byte buffers for intermediate storage of 343 00XXXXXX words before writing or after reading. DIIDD, on the other hand uses only an 86-byte **auxiliary buffer** in addition to the 256-byte memory area that is specified as the read destination or write source. In **PRENIBBLIZE** ($FDF0), the lower two bits of every word are packed into the auxiliary buffer in the XXXXXX00 format. The auxiliary buffer and the unmodified 256-byte memory buffer then become the source for the WRITE DATA FIELD ROUTINE. During writing, the data from the 256-byte buffer is converted to the XXXXXX00 format via "AND #$FC" instructions ($FD56, $FD84, $FD9B).

Apple's purpose in eliminating the 256-byte intermediate buffer in DIIDD was not to save space. It was, rather, to save the time it takes, in prenibblization, to strip off the two LSBs and store the 256 XXXXXX00 words in the intermediate buffer. Packing the two LSBs of 256 bytes into 86 XXXXXX00 words in the auxiliary buffer still takes a lot of time, but the code that does this ($FDF0—$FE43) is optimized for speed so PRENIBBLIZE execution time is minimized.

A lot of complicated code was devised just to speed up prenibblization, but this goal was nicely achieved

*The ProDOS speed improvement also stems from the fact that one call to DIIDD results in two sector accesses, from sector interleaving designed to take advantage of DIIDD improvements, and from streamlined MLI file handling including a "direct read" mode that transfers file read data directly from a disk to its memory destination without first transferring it to a ProDOS file buffer.

since DIIDD prenibblization takes only about 6,450 cycles as opposed to about 10,950 cycles in RWTS. The savings of 4,500 cycles represents more than one third of a sector (one sector passes the drive head in about 12,750 cycles). To put it another way, with every other sector interleaving, ProDOS can use about 5800 cycles (12750 − 6450 − 500 cycles for slow DIIDD head positioning) of processing time between Command 2 calls to DIIDD. If DIIDD prenibblization took 10,950 cycles, ProDOS would be able to use only about 1300 cycles between Command 2 DIIDD calls.

Apple was able to achieve an even better performance improvement in Command 1 DIIDD processing. Whereas **postnibblization** in RWTS takes about 10,250 cycles and occurs entirely after the data has been read from disk, postnibblization in DIIDD is performed on the fly (while the data is being read) and adds no processing time to Command 1 DIIDD calls. In other words, ProDOS can process data for up to 12,750 cycles between Command 1 DIIDD calls, and still pick up the next block without waiting for a complete disk revolution.

The **READ DATA FIELD** subroutine, which is called after the address field for a specified sector has been read, begins at $FBFD. At entry to this subroutine, the read syncing leader preceding the data field will be passing the drive head, so there is some time for processing before the $D5 $AA $AD data field identifier is expected. This time is utilized to modify some instruction operands in the subroutine so disk data can be read via "LDX $C0nC" instead of "LDA $C08C,X" as is required by the reading scheme, and so the bottom third ($FC6F), middle third ($FC96), and top third ($FCAE) of the 256-byte buffer can be accessed via "STA $nnnn,Y", thus saving a cycle over the "STA (ZP),Y" alternative.

After the pre-read processing, the data field identifier is read ($FC31), a running checksum is initialized ($FC55), and 86 disk words are read ($FC59), decoded to XXXXXX00 format ($FC5E), and stored in the auxiliary buffer ($FC61).

Next, the bottom third ($FC69—$FC83), middle third ($FC84—$FC9A), and top third ($FC9C—$FCB7) of the 256-byte buffer are read from disk and decoded to XXXXXX00 format, then ORed with 000000XX data which is postnibblized on the fly from the auxiliary buffer XXXXXX00 data. The combined XXXXXXXX data which is stored to the 256-byte buffer is true 8-bit data, just as it resided in a 256-byte buffer before it was stored on disk by a Command 2 call to DIIDD.

Because of the extra processing that takes place between reading of disk data words, DIIDD reading loops take longer than RWTS reading loops. As opposed to the 26-cycle reading loops of RWTS, the reading loops of DIIDD take 26 (auxiliary buffer), 29 (bottom, middle thirds), or 28 (top third) cycles. This means that DIIDD will not tolerate as fast a reading drive (or slow a writing drive) as RWTS will, and it indicates why the FILER formatting routine has such tight specifications on the length of the address field read syncing leader and, consequently, on drive speed.

There is a particularly tight spot between reading the bottom third and middle third of the 256-byte buffer where there are 30 cycles between data register polling ($FC75—$FC88). I believe that this single instruction sequence determines the drive speed variation which DIIDD will tolerate since it is the weak link in the chain. Given that data is written in 32-cycle loops and that the disk data word should be held valid in the data register for one cycle more than a 7-cycle polling loop (see Table 9.5), the 30-cycle period between reads means that there can be no more than 3 parts in 32 variation in speed between the reading and writing drives. In other words, drive speed variation plus speed variation due to disk flutter must be less than plus or minus 4.7% or occasional reading errors will occur at the bottom/middle border in a data field.

I believe that Apple could make this speed tolerence less critical with some minor modifications to DIIDD. The modification would center around moving the auxiliary buffer from $FB00—$FB55 to $FBAA—$FBFF. This would change the instruction at $FC7A from "LDX $FA56,Y" to "LDX $FB00,Y" and, because the Y-indexing would not be across a page boundary, reduce the 30-cycle delay to 29 cycles. The change would require the retuning of the writing loop at $FD3A—$FD4E, but this could easily be accomplished by getting slot number via absolute addressing at $FD44. Additionally, all code from $FBFD on would have to be moved back three memory locations. No problem, right? I hope not. The person who wrote this code was so good that it scares me to second guess him in print. Nonetheless, maybe he missed just one little trick.

HARDWARE APPLICATION

INSTALLING A WRITE PROTECT SWITCH ON THE DISK II DRIVE

Did you ever want to store information on a write protected diskette? This involves removing the write protect tab or cutting a write protect notch, writing the file, then sticking on a new write protect tab. Here's another one. Did you ever delete a file, then immediately regret it? If you haven't run into one of these situations, then I'll only say that your ball sure bounces more nicely than mine does. This application note details a simple modification to the Disk II drive which enables you to write on write protected disks and also gives you time to have second thoughts about writing files to or deleting files from a disk.

The modification involves installing a single switch on the front of the Disk II drive. The 3-position switch allows selection between normal operation, forced write protection, or bypass of write protection. If you normally leave the switch in the protect position, you will always have to take an extra step to write, delete or rename a file. This is a fairly normal feature of expensive storage peripherals associated with mainframe computers. Normally, taking the extra step required to overwrite data makes the operator think twice about possible destruction of important data.

The bypass position of the switch allows you to write on a protected disk. This might be of use in writing on disks you have protected for your own reasons, writing on special diskettes that have no notch, and writing on the backs of single sided disks if you do that sort of thing. If you happen to be a software publisher, this mod is a must.

The idea of the write protect switch is not my own. It was pointed out to me that the modification had been suggested in magazine articles. Once it is realized that such a modification is possible, the design of the modification is fairly obvious.

Figure 9.21 shows the modification. What is involved is installation of the new switch and rewiring of the already present switch which is activated by the write protect notch. The type of switch required is a ON-OFF-ON SPDT (Single Pole, Double Throw) switch. The installation procedure given here involves installing it on the front of the drive, but you may prefer to put the switch on a remote box. When buying the switch, select one that switches very easily so a minimum of stress is placed on the plastic front panel of the drive. Since the notch activated switch is mounted below the disk slot, the new switch should also be mounted below the slot so wires will not interfere with disk insertion. Choose a mounting point near a reinforcing structure on the back of the front panel to give added strength to your installation. All wiring should be soldered, and 24-gauge insulated wire works nicely.

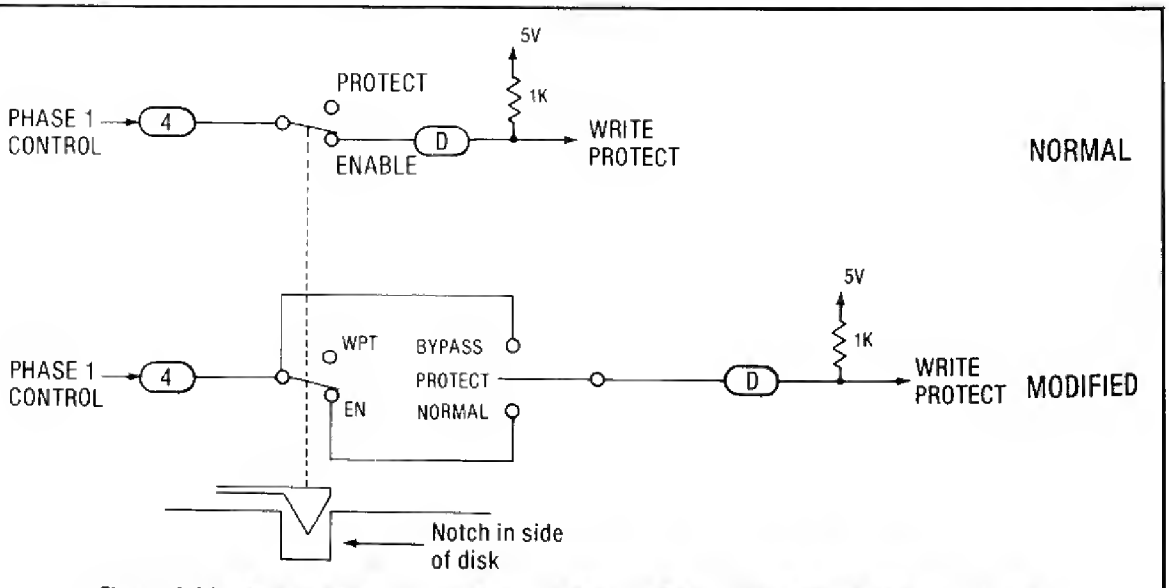


Figure 9.21 A Single Switch on the Front of the Drive Allows the User to Select Normal, Forced Write Protect, or Bypass Write Protect.

The purpose of steps 5,7, and 11 is to improve accessibility, and you may elect not to perform them if you are good at working in tight spaces.

Installation Procedure:*

1. Turn off the computer and remove the controller from its slot. Mark pin 1 of the 20-pin ribbon cable connector and plug with fingernail polish so you will not reinstall the cable incorrectly. Disconnect the ribbon cable from the controller and move the drive to a convenient work area.
2. Remove the four screws from the bottom of the drive. Remove the white case by holding the drive on your palm and sliding the case to the rear.
3. The notch activated switch can be seen toward the front on the left side of the drive. Select the location of your ON-OFF-ON switch so that wires can be connected between the two switches in a way that drive mechanisms are not interfered with. Hold the switch near the selected spot to make sure no problems will arise. The location shown in Figure 9.22 works well with a small bodied switch.
4. Mark and drill a hole for your switch. Clean out any plastic filings which fall into the drive.
5. The big horizontally mounted card is the analog card. You may remove it by disconnecting the two plugs at the back and by removing the 4-pin read/write head plug. Remove the retaining screw on either side, and the analog card slides out. You may wish to clean the head with alcohol and cotton swabs at this point.
6. Remove the two beveled head machine screws from each side of the front panel. Position the panel to the side, attempting not to strain the wires connected to the IN USE indicator.
7. Use a fine lead pencil to outline the position of the notch activated switch. This way you can reinstall it in the same position. It will also help to slide a disk in and out of the drive while you observe the switch action, so you will be able to reproduce the same action at reinstallation. Remove the two allen screws which hold the switch to the side of the drive.
8. A black wire is connected to the normally closed contact of the notch activated switch. Desolder the black wire. This wire needs to be connected to the ON-OFF-ON switch and it will probably require a short splice. Splice a short jumper between the black wire and the center or common contact of the ON-OFF-ON switch. Solder both connections and insulate the splice connection with electrical tape or by another suitable method.
9. Connect a wire between the desoldered terminal of the notch activated switch and the NORMAL mode contact of your ON-OFF-ON switch. If you choose to have NORMAL mode in the down position, then the upper contact will be the NORMAL mode contact, and vice versa.
10. A brown wire is connected to the common contact of the notch activated switch. Solder one end of a jumper wire to the same contact as the brown wire. Solder the other end to the BYPASS mode contact of your ON-OFF-ON switch. The BYPASS mode contact will be opposite the NORMAL mode contact.
11. Remount the notch activated switch to the side of the drive, aligning it to the outline you drew with a pencil. Slide a disk in and verify that the switch clicks on and off as the disk notch is engaged and disengaged.
12. Mount the ON-OFF-ON switch to the front panel.
13. Reinstall the front panel on the drive, making sure the spindle engagement mechanism mates with the grooves on the hinged door on the front panel.
14. The remainder of reinstallation is the reverse of the dismantling steps. You can easily verify operation by attempting to delete some test files and observing the write protect indication. Do not operate with any disk containing important files until you have verified correct operation of the modified drive.

Figure 9.22
A Drive With the Write Protect Switch Installed.

*Please read the NOTE OF CAUTION at the beginning of the book before making any modification to your hardware.

You should mark the functions of the new switch on the front panel. I used white dry transfer letters (available in electronic stores) for this purpose. Afterwards, I sprayed the switch area with an acrylic coating (also available in electronic stores) to protect the lettering. All drive openings should be covered when spraying with acrylic coating to prevent accidental coating of the read/write head.

Some alternate source drives for the Apple use a light emitter and photo switch to check for write protection. Rewiring the photo switch in conjunction with a new ON-OFF-ON switch should also be possible on most or all such drives. I happen to own a FOURTH DIMENSION drive manufactured by Siemens. I added the write protect switch mod to it by splicing into the wires going to the photo switch. On the 4D drive, the photo switch is mounted over the disk slot. The white wire is the input to the photo switch and is the equivalent of the brown wire in the above procedure. The yellow wire is the output of the photo switch and is the equivalent of the black wire in the above procedure.

chapter 10

Maintenance and Care of the Apple IIe

The modern microcomputer is such a marvelous thing. Just think of the accumulated knowledge and industrial capability of the human race represented by such a machine. Invented by man, feared by man, exploited by man, and hated by man. Especially hated by man when it doesn't work right. After years of having computer systems subjecting us to impersonal and illogical errors, we have advanced to the point where we have computer systems in our own homes subjecting us to personal and illogical errors.

The vast majority of computer mistakes are caused by imperfect programs. The more involved a program, the greater the chance of an oversight by the programmer. We used to curse the computers. Now, when a husband writes a program that allows his wife to enter her kitchen recipes, then destroys them in milliseconds, it's not the computer that gets cursed. Yes, today's computers are very personal.

Occasionally, computer malfunctions will actually be caused by a hardware failure rather than a program in disarray. This should be a fairly rare occurrence, because digital electronic circuitry is so reliable. Yet, hardware failures do occur, and most of us encounter them eventually in our home or business system. In this chapter, attention will be given to the maintenance philosophy of the Apple IIe computer. There will be some discussion of what options you have when your system fails, and of some simple fault isolation steps which can be taken by you in your home. We also will discuss ways of reducing the probability of hardware failure in your system.

APPLE HARDWARE RELIABILITY

There is no electronic circuitry more reliable than modern digital electronic circuitry. Digital ICs routinely operate for thousands of hours without failure, and they are easy to replace if they do fail. The Apple computer is consequently a very reliable machine. There are, however, less reliable facets of a computer than the ICs that populate it. Some weak links in the reliability chain are discussed here.

Tinkering Users

If you are very involved with your hardware, trying new and different things all the time, you are bound to make some mistakes which cause hardware casualties. People who like to tinker with their computer should do so, because it's as fun as all get out. Those same people would be naive to think they may not occasionally mess something up. Even though my wife thinks otherwise, I have probably set no records in this area. I do, however, consider myself an Apple clobberer of the first degree. The personality traits necessary to reach this plateau of destructive potential are an infantile curiosity and terminal absentmindedness.

The possibility of causing hardware failure by tinkering leads to the following common sense rule. If your Apple is your bread and butter—if it costs you money when it's not running—don't mess with it. If your Apple is your creative outlet, then play with your toy any way you please.

The Peripheral Slots and Auxiliary Slot

It is this author's opinion that the most important hardware feature contributing to the Apple's marketing success is the concept of peripheral slots, mounted on the motherboard with address decoded control signals generated on the motherboard. The peripheral slots do represent a reliability weak point though. Suppose you had a television which allowed the owner to enhance it in all sorts of ways by lifting the lid on the television and installing cards in slots on a big motherboard. Surely, every owner who reconfigured his television a lot would mess it up eventually, and the TV repair industry would be happy with the extra business.

The sort of thing that can go wrong when pulling or removing cards is that a card might get installed backwards. I've only done this twice. It tends to wipe out one or more chips on the card. As soon as a chip shorts a power supply voltage to ground, the power supply shuts itself off and damage to further chips is prevented.

An impatient owner might remove or install a card with power applied to the Apple. You can usually get away with this, but when you see a spark, cross your fingers. I've never burned up more than one chip at a time doing this, but it's possible to wipe out every chip connected to D0 of the data bus. This is because D0 is adjacent to +12 volts on the peripheral slot pins and they may get shorted together. Combining two reliability hazards, you come up with the most common cause of hardware failure in the Apple, a tinkering owner who installs and removes cards while power is applied.

One of the worst things that can happen while installing or removing a peripheral card is that flexing the motherboard might cause a hairline fracture of a current trace or solder joint. Of course the same thing could happen if you drop your Apple or if a manufacturing defect starts to show symptoms. The resulting marginal electrical contact can cause system problems to come and go in a random way based on such variables as temperature, motherboard stress, and the price of hogs in Kansas City. A good computer technician might be able to isolate a problem like this on a good day if you can afford the wages of a good technician for a whole day.

A reinforcing bar mounted at the back of the Apple IIe reduces motherboard flexing to the point where occasional card insertion and removal is not likely to overly stress the motherboard. Even with the reinforcing bar, however, the mechanical integrity of the motherboard mounting is marginal. Persons who regularly remove and insert peripheral cards should, therefore, not be surprised if problems related to mechanical stress eventually arise. Such problems are much more likely if an auxiliary card is removed and installed regularly, because the mounting near the auxiliary slot is extremely flimsy.

Peripherals with Moving Parts

Moving parts are a reliability problem in any industrial creation. Compare an automobile to a computer. The automobile might run 100,000 miles before its effective life is over. This will be 5000 operating hours at 20 miles per hour with many parts replaced along the way. Yet you could turn on your Apple and let it run for 208 days in a row (5000 hours) and have a very reasonable chance of experiencing no hardware failure. If a part fails, you can replace it and go another few thousand hours without a failure. The main limiting factor on effective life is obsolescence.

Now take a disk drive, an electro-mechanical device with precise mechanical alignment. Don't expect to run a disk drive for 208 solid days without hardware failure. Friction can cause the motors or head assembly to wear out, and like the front end alignment of a car, drive alignment sometimes goes out. Cleanliness becomes a factor. If you are really putting a lot of hours on your Apple, then you should expect to eventually have to have some disk drive maintenance performed. The same is true of a printer. Heavy usage results in wear on the moving parts

and in probable eventual maintenance requirements. This is why printer manufacturers advertise how few moving parts are in their products.

The Power Supply

In the hypothetical 208-day reliability test that was mentioned earlier, if any unit failed, it would most likely be the power supply. This is because the electric currents in the power supply are so much greater than in any IC. Of course the power supply is rated to handle a lot of current, but high current devices are usually more apt to fail than low current devices. Also, if the AC line voltage fluctuates, the power supply is the unit most likely to be damaged by the resulting current surges.

Application and removal of power to the Apple can be thought of as a controlled fluctuation. The ICs and power supply are designed to handle the current surge that occurs when the switch is turned on. Still, there is no time when your computer is more likely to fail than when the power is fluctuating, including when you turn the power switch on. This means you should turn the Apple off if power starts fluctuating, as when the lights in the house go dim during a storm. It also means that you should not needlessly turn the power to the Apple off and on.

A particular reliability problem with the Apple power supply is the power switch. The switch arcs sometimes at power up, and this can eventually cause the switch to malfunction. Amazingly, this problem has been ignored for years by Apple, even though the rest of the world acknowledges its existence by using an external switch to turn the Apple on and off. The part is a turkey, and it should have been replaced by now in the Apple IIe power supply design with one that will last.

IMPROVING YOUR APPLE'S RELIABILITY

The reliability weak links give some hints on how to improve the reliability of your own Apple:

1. Above all else, never remove or install peripheral cards, the auxiliary card, or ICs with power applied.
2. To improve reliability, don't tinker with the Apple or peripherals. This must be each person's compromise between the conflicting desires of wanting a reliable computer and wanting to tinker. You probably know that my personal choice is to tinker all I want.
3. Keep the Apple covered when not in use so that the electronics stay clean and nothing is accidentally dropped inside. Don't set coffee or sodas on the Apple, because you may spill them.
4. Don't operate the Apple on an unstable power source. Be wary of operating during electrical storms because power may fail.
5. Connect power to the Apple through a bus bar or other device with a switch on it and turn the Apple on and off using this switch to save wear on the power supply switch. The bus bar may have current surge suppressors built-in which help stabilize the power applied.
6. You may elect to reduce the operating temperature of the Apple by mounting a fan on the case.

Products are available which perform the three tasks of providing an external switch, surge suppression, and temperature reduction. None of these is necessary for operation of the Apple, but it can be argued that each one could improve its reliability. Necessity of temperature reduction is the most questionable. The Apple uses commercial grade (as opposed to military grade) components which are guaranteed to operate within specifications over the 0—70 degrees Centigrade (32—158 degrees Fahrenheit) operating range. My measurements of the Apple IIe operating temperature indicate that it is under 130 degrees Fahrenheit just above the 6502, which is the hottest spot I could find. This is well within the 158-degree specification of the components. Reducing this operating temperature should still reduce thermal expansion, reduce the possibility of malfunctioning of components that are not up to specifications, and reduce the possibility of malfunctioning due to overloading of signals by too many peripheral slot cards. Also, by reducing power supply temperature, one would expect to increase the amount of current that can be supplied before overheating and failure of power supply components occurs.

To see for myself the effect of using a fan on the operating temperature of the Apple IIe, I ran a 4-hour test, measuring the temperature at the top of the power supply and just above the MPU with and without a fan running. The test is patterned after a similar test I performed on an Apple II for *Understanding the Apple II*. The fan used was a Super Fan II made for the Apple by R. H. Electronics. It hangs from the left side of the case and has an external on/off switch and surge suppression. Four peripheral cards were installed in the Apple under test, which was a Revision B Apple IIe. The temperature measuring device was a pyrometer which utilizes a thermocouple for a probe.

The results of the test are shown in the graph in Figure 10.1. As the graph indicates, it gets very warm near the surface of the MPU. Also, the fan reduces the operating temperature by about 10—15 degrees Fahrenheit. Please note that the graph is probably a good indication of the relative temperatures with the fan off and on, but that there are many variables which affect the absolute reading, including the temperature of the room, probe placement, and instrument accuracy. Additionally, the test does not measure the amount of improved component heat dissipation due to forced air movement in the cabinet. The measurements made at the surface of the power supply were probably closest to the ambient air temperature in an Apple IIe. It would be fairly accurate to say that the ambient air temperature in the Apple IIe cabinet is about 25 degrees F. greater than room temperature, that the temperature differential can be reduced by about 15 degrees F. with a fan, and that some improvement in component heat dissipation will occur. Whether or not this is worth the price of a fan is a subjective matter.

REPAIR OF THE APPLE IIe

Repair of a broken digital computer is different than repair of other sorts of electronic equipment. Many uncertainties of electronic circuit operation do not exist in digital equipment, because most of the circuitry is made up of 2-state electronic switches. In most circuit elements, either current flows or it doesn't flow. This is a simple condition compared to the infinite variety of signal conditions which exist in analog electronics.

The complexity of digital equipment lies not in complex electronics but in complex logical capabilities. As a result, any difficulty in the repair task will often be due to logical complexity. This is good,

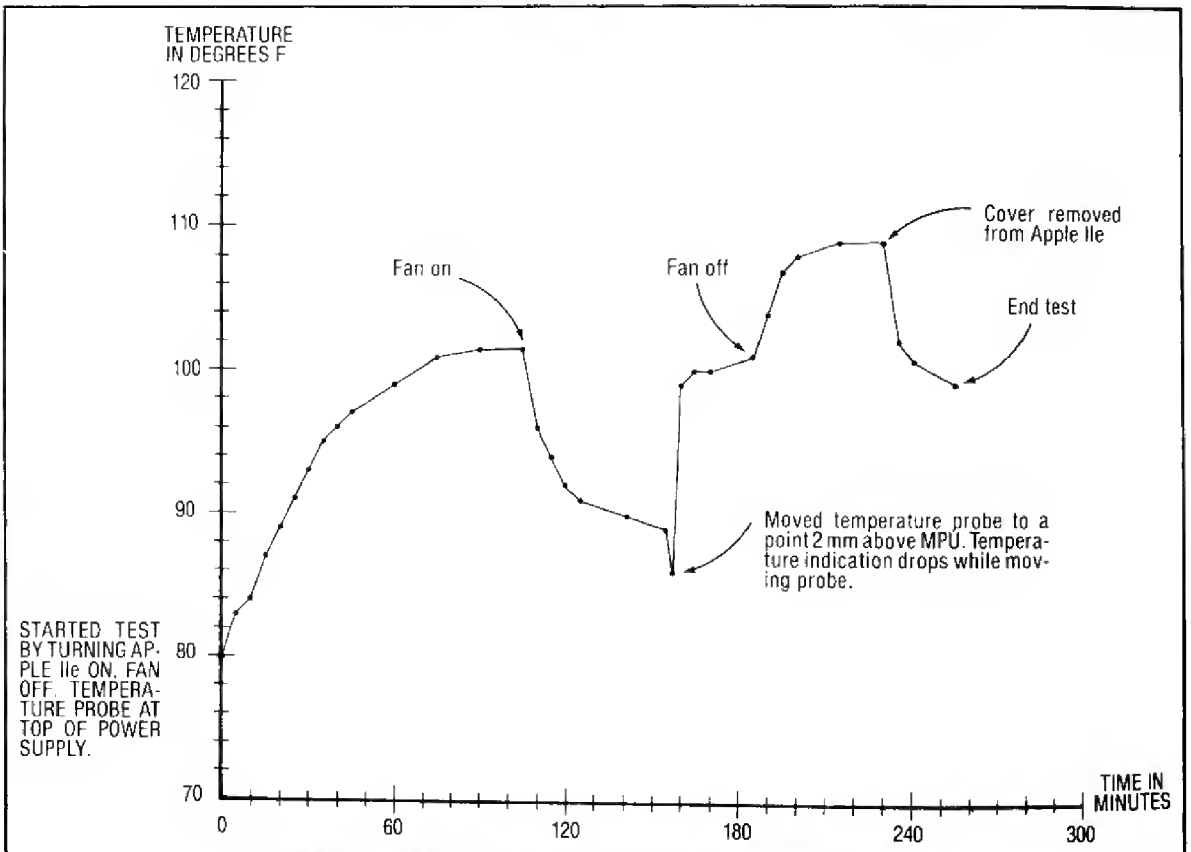


Figure 10.1 Temperature Measurements Inside an Apple IIe.

because it means many hardware functions in a computer can be verified, and many casualties can be isolated by self diagnostic programs. More troublesome problems can be diagnosed by external computers designed and programmed to troubleshoot certain classes of problems. Since computer casualties are often logical malfunctions, what better way is there to solve them than by logical analysis using a computer?

The Apple IIe has a modest end user self-diagnostic capability consisting of motherboard firmware routines that check motherboard RAM, ROM, and IOU and MMU flag manipulation and reading. However, these firmware diagnostics are somewhat limited in value since the Apple must be pretty healthy before they can even be executed. There is no hardware or firmware based timing or video signal verification, although auxiliary cards which perform these tasks must certainly exist in Apple's checkout and repair facilities. RAM based diagnostic programs are available on disks to computer dealers but are not available to owners. There is, therefore, very little an owner can do by way of isolating difficult casualties unless there is an in-house computer technician with some test equipment. There are some checks that can be made by anybody, but more on that later.

The typical computer retailer will have what Apple calls a Level I repair capability. They will often have a shop and a computer technician. They will also have a diagnostic peripheral card that is made by Apple, several disks full of diagnostic programs, and a parts and information pipeline to Apple. The diagnostic peripheral card contains firmware to check the operation of several areas including RAM, ROM, the keyboard, and the video display. If their disk-based diagnostics will load, they will verify and isolate faults in any peripherals for which they have diagnostics. The technician or salesman can then replace components that are indicated to be bad and hopefully fix the malfunctioning Apple very cheaply. The dealers also have disk drive alignment disks and procedures which allow their technicians to precisely align the Disk II drive and some drives that are compatible with the Disk II.

Dealerships with more sophisticated repair capabilities will also work on some products for which they have no advanced diagnostic aids, such as printers or modems. If there is a resident Apple technician with an oscilloscope, he probably will repair many problems not pointed out or isolated by the diagnostic aids.

When a problem is beyond the capability of a dealership or the repair of a problem will be so time consuming that it will not be cost effective for the dealer or customer, the dealer will swap out a major assembly such as a motherboard or power supply for a very reasonable cost. He then sends your repairable assembly to an Apple Level II repair facility. Apple can repair the assembly more easily because they have sophisticated test fixtures, documentation, and assets not found in a computer dealership. Other companies besides Apple will also have a turn around policy on their Apple compatible products. Therefore, when a peripheral card fails, you may well be able to get a quick swap at a computer dealership.

Apple will not allow Level I repair shops to perform some tasks. They maintain this control by refusing to swap assemblies upon which unauthorized work has been performed. For example, a Level I shop can change the analog card in a disk drive, but they can't change the drive motor. Also, with few exceptions, Apple won't accept a modified assembly. Apple's no swap rule for modified assemblies is fairly reasonable considering that they must make their own repair operation cost effective.

The primary hardware based self diagnostic features of the Apple IIe are the peripheral slots and the auxiliary slot. An empty slot becomes a diagnostic port when you plug a specially designed test fixture into it. There can be no doubt that Apple must use such test fixtures for production check out as well as fault isolation, because they are an obvious necessity. From a peripheral slot, you can verify power supply voltages, verify some timing signals, check RAM, ROM, the MMU, the IOU, and all address bus command features via DMA, measure for shorts at all pins, exercise the 6502 with a test program via the INHIBIT' line, and verify correct interrupt response. From the auxiliary slot, you can monitor timing, video generation, and memory management signals, and inject substitute timing signals or a substitute picture signal. Many problems with individual ICs can be isolated to the chip while other problems may be isolated only to an area. Further isolation of problems can be performed by attaching jumpers from smart peripheral or auxiliary slot test fixtures to various ICs. Apple probably has an area full of engineers who do nothing but design test fixtures for Apple products, program them, and write test procedures for them.

We pay for this diagnostic capability when we buy Apple products, even though it isn't built into the Apple IIe. Large scale automated checkout and

fault isolation of a product is the only way to provide quality assurance and service a complex mass production device in a cost effective way. Money for developing this capability must come from sales and service revenues.

WHEN YOUR APPLE BREAKS

When your Apple breaks, chances are that you will have to take it to a computer dealer for repair. Yet, there are some very simple checks you can make which might get your Apple up in a hurry. These are checks which can be made by anyone, and they are the kind of checks a salesman might make if you brought your Apple in on the service technician's day off. Be aware that **any damage you cause while making these checks will void your warranty if it is still in effect and make out of warranty repairs more expensive.** Also, any use of test equipment by unauthorized service personnel might put your warranty in jeopardy.

In these discussions, the use of a multimeter, logic probe, or oscilloscope will be occasionally called for. If you do not have access to the instrument mentioned, or you do not know how to use it, it is time to get your system to a dealer. Incidentally, you can buy a logic probe and a multimeter for $20 each at **Radio Shack** and learn how to use them in a few minutes.

If you manage to find the exact cause of a problem, you can buy most Apple components in computer electronics stores or, more expensively, at computer dealers. The IOU and MMU, of course, can only be purchased at an Apple dealership or directly from Apple Computer, Inc. The computer dealer will charge you more than an electronics store for parts, because he is not in business to sell electronic components. Like virtually all American maintenence operations, the computer dealer will put a big markup on his parts prices to improve the profitablity of his service department.

There are hazardous voltages inside the Appl but they are all in the power supply. Nevertheless, is a good idea to pull the plug on the Apple anytim you are working inside. Never work on the powe supply with the line cord attached. Many of th power supply components are not isolated from th line voltage and there are numerous dangerous voli age points in the power supply.

The Firmware Diagnostics

The firmware diagnostics at $C400—$C7FF* pro vide the Apple IIe end user with a modest capabilit for verifying correct hardware operation. They als are designed to indicate bad RAM chips, bad ROM chips, and MMUs or IOUs whose soft switches don' operate correctly, but because of poor overall con cept, it is unlikely that the firmware diagnostics wil correctly indicate a bad ROM or RAM chip. Th reason for this is that the diagnostics will not run i there is a completely bad ROM or RAM chip.

When an operator performs a close Apple reset o a functional Apple IIe, the reset routine at $FA62 o the motherboard E0—FF ROM is executed. Severa subroutines are immediately executed via JSR in structions, and then flow is passed to $C100 via th GOTOCX subroutine. If at $C23F, close Apple is found to be held down, flow will then be passed to th diagnostic routines at $C401. The contents of th diagnostic routines are as follows:

1. **Initialization** at $C401 includes saving open Apple key status, stack pointer = $FF, CSW = COUT1, disabling high RAM for reading and writing, and calling the HOME subroutine.
2. An **MMU soft switch** check at $C41D consists of reading (at $C011—$C018) and verifying the

*The firmware diagnostics are described here as they are in the original Apple IIe firmware. Please note that, in the enhanced firmware, the diagnostics reside at $C600—$C7FF, there is no ROM check, and both motherboard and auxiliary card RAM are checked. See Chapter 6, **The Apple IIe Firmware Upgrade** for a description of the enhanced firmware.

Table 10.1 MMU Soft Switch Diagnostic Error Codes.

SOFT SWITCH	INITIAL STATUS	CHANGED STATUS
BANK1	0	8
HRAMRD	1	9
RAMRD	2	A
RAMWRT	3	B
INTCXROM	4	—
ALTZP	5	C
SLOTC3ROM	6	D
80STORE	7	E

diagnostic routine input status of the MMU soft switches. Then the state of the MMU soft switches excluding INTCXROM is changed and the $C011—$C014 and $C016—$C018 status is verified. If any errors are detected, the message "MMU FLAG E4:a b . . . n" is printed where (a b . . . n) are error codes indicating errors as shown in Table 10.1.

3. An **IOU soft switch** check at $C498 consists of reading (at $C01A—$C01F) and verifying the diagnostic routine input status of the IOU soft switches. Then the state of the IOU soft switches is changed and the $C01A—$C01F status is verified. If any errors are detected, the message "IOU FLAG E5:a b . . . n" is printed where (a b . . . n) are error codes indicating errors as shown in Table 10.2.
4. After the IOU soft switch check, the program hangs at $C510 with error messages displayed if there was an MMU or IOU error. If there was no MMU or IOU error, the C1—DF ROM check is performed.
5. The **C1—DF ROM** check at $C53F consists of summing $C100—$DFFF excluding $C400—$C4FF and comparing the result to the contents of $C400. If the sum is wrong, the message "ROM:E8" is printed and the program hangs at $C569. Otherwise, the E0—FF ROM check is performed.
6. The **E0—FF ROM** check at $C56C consists of summing $E000—$FFFF excluding $F7FF and comparing the result to the contents of $F7FF. If the sum is wrong, the message "ROM:E10" is printed and the program hangs at $C58E. Otherwise, the RAM check is begun.
7. The **motherboard RAM** check at $C591 begins, after a short initialization, by filling all motherboard RAM, but not auxiliary card RAM, with $00 ($C5A8). During this initial zero fill, the speaker will toggle at about 180 Hz if open Apple was held down when the diagnostics were initialized.
8. RAM is checked and toggled ascending ($C5EB), and again ascending ($C64A), and checked and toggled descending ($C64C), and again descending ($C6AD). During these checks, the speaker is toggled at about 10,000 Hz if open Apple was held down when the diagnostics were initialized. If an incorrect value is read at anytime during these checks, the message "RAM:Fa Fb. . . Fn where (Fa Fb . . . Fn) are those RAM chips of RAM chips F13—F6 yielding incorrect data (example: RAM:F13 F8 F6). The system then hangs at $C714 with the RAM error message displayed. Memory location $01 contains the memory page in which the error was detected.
9. If no errors are detected and open Apple and close Apple are not both held down at the completion of diagnostics, the message "KERNEL OK" is displayed, then the program hangs at $C72D. "KERNEL OK" means the diagnostics passed, that ROM, RAM, and the soft switches are good, and that the programmer who wrote the diagnostics was more interested in useless jargon than in a meaningful English language message.
10. If open Apple and close Apple are both held down when the RAM diagnostics are completed without error ($C720), an **I/O exercising program** is moved from $C7AE—$C7E2 of ROM to $0100—$0134 of RAM and executed. This program enables slot I/O ($C7AE), makes ascending read and write references to $C090—$CFFF ($C7B6), makes descending read references to $C079—$C000 ($C7CF), then disables slot I/O and recycles the diagnostics ($C7DA).

Some aspects of interpreting diagnostic performance are obvious, but others are not. Here are a few notes about diagnostic interpretation:

1. If one of the ROM chips is bad, the portion of reset routine beginning at either $FA62 or $C100 will not be fetched and executed. If a RAM chip is bad, then the flow will not return to the reset handler after subroutines are executed because the return link information will not be correctly saved and retrieved from the stack. In

Table 10.2 IOU Soft Switch Diagnostic Error Codes.

SOFT SWITCH	INITIAL STATUS	CHANGED STATUS
TEXT	0	8
MIXED	1	A
PAGE2	2	9
HIRES	3	B
ALTCHRSET	4	7
80COL	5	6

other words, if a RAM or ROM chip is completely bad, flow will never arrive at $C401 and the diagnostics will not be executed.

2. There are four major checks performed in the following order: soft switches, C1—DF ROM, E0—FF ROM, RAM. If any of these checks fails, an error condition is indicated and the remaining checks are not performed.
3. Memory locations in Page 0 of RAM are utilized in performance of all checks, and error printing subroutines are called via JSR instructions. Therefore, if the diagnostics were somehow performed with Page 0 or Page 1 malfunctioning, performance would be unreliable.
4. The RAM test audio tones provide a means of verifying diagnostic performance in the absence of video display. This is true, even when the speaker is disconnected, because the LED connected across the speaker jack will glow during the 10000 Hz toggling if the speaker is disconnected.
5. An intermittent error might show up only if the diagnostics are recycled many times. To recycle the diagnostics, turn off the Apple IIe and disconnect the keyboard plug, paddles or joysticks with pulled down pushbuttons, and any peripheral cards that would interfere with recycling. Remove the disks from all of your disk drives* and turn on the Apple IIe. The diagnostics, including the portion that exercises I/O, will recycle until an error occurs or you turn off the Apple IIe.

Since operation of the diagnostics is normally precluded by bad RAM or ROM, the diagnostics are good for verifying good memory, not troubleshooting bad memory. Properly conceived Apple IIe diagnostics would reside solely in the E0—FF ROM. In this ROM, the first action taken at any reset would be a BELL routine which did not depend on RAM to operate correctly. The ability to execute a stored sequential program would thus be verified independently of RAM. If the BELL does not ring, RAM and the C1—DF ROM have nothing to do with it.

After ringing the BELL, the next step would be to check the close Apple button to see if diagnostic execution is selected by the operator. If diagnostic execution is selected, the first task is to verify RAM Page 1 and Page 2, again in a routine that does not utilize RAM except to verify it. This routine should be executed many times to absolutely verify the reliability of these critical pages. Failure of the Page 1/Page 2 verification should be indicated by speaker beeps, and passage would enable diagnostic continuation with knowledge that indirect addressing and the stack can be used.

Once Page 1 and Page 2 of RAM are verified, the programmer has freedom to make all sorts of checks including complete RAM, ROM, MMU, and IOU checks. Complete diagnostics would certainly verify auxiliary card RAM, the MMU memory configuration capabilities, and the existence of 17030 cycles in VBL. All pass/fail indications should be via speaker beeps (Morse code?) in addition to video display so the diagnostics could be interpreted when the display was malfunctioning. A diagnostic E0—FF chip like this would isolate some problems that might occur in an Apple IIe. At least they would give you a decent indication before you warmed up the oscilloscope.

The Peripheral Card Check

A check that should be made at an early point for almost any persistent symptom is to turn off the Apple, remove the auxiliary card and all peripheral cards, turn the Apple on, and see if the symptom disappears. If it does, you can find which card is causing the fault by turning the computer on with each card installed by itself. Then you can operate the Apple, losing only the capabilities represented by the malfunctioning card until it is repaired. Even when a peripheral card or auxiliary card is known to be malfunctioning, it is a good idea to check operation with all other cards removed. For example, your disk controller may be loading down the RESET' line and causing the printer card to misbehave. This procedure of isolating a problem to a peripheral card will be referred to as the **peripheral card check**. Other than this most basic of checks, your course of action will depend on your symptoms. The most easily recognized symptom is a completely dead Apple, normally indicating a power supply problem.

Power Supply Problems

There are two symptoms you will normally encounter with power supply problems. The Apple is dead and there is a low level clicking noise coming from the power supply, or the Apple is dead and silent too. If there is a clicking noise, the power

*Increasing references through the disk controller DEVICE SELECT' addresses leave all stepper phases on and drive 2 on and configured for writing. This does not wipe out the disk in drive 2 if phase 1 on forces write protection in your drive like it does in the Disk II drive. Prudence dictates that you remove all disks before recycling the diagnostics.

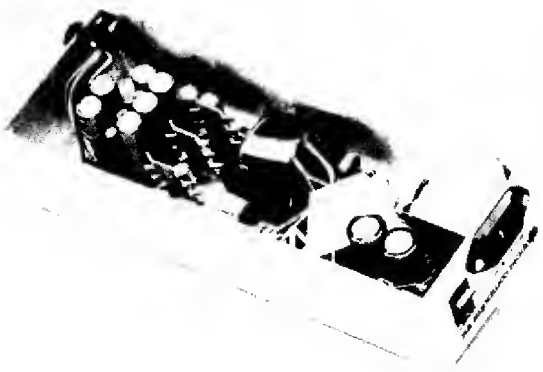


Figure 10.2 A Power Supply With the Bottom Off.

supply is quite likely good, but a motherboard, peripheral card, or auxiliary card malfunction is causing an overload condition. If the Apple is dead silent, there may be a casualty in the power supply itself.

The clicking noise is the tinkerer's symptom. Chances are very good that somebody was installing, removing, or modifying something in the Apple. When any casualty symptom follows tinkering, concentrate your investigations in the areas that were tinkered with. If something was touched, look it over.

When the clicking noise occurs, turn the computer off immediately. If a component has just shorted, it may be hot or show evidence of burning. Visually inspect the motherboard and peripheral cards under good lighting. Look for such things as ash or black marks on the components. Touch all the ICs lightly to see if any are hot. Be careful or you might burn your finger when you get to the right one. Check that all cards, plugs, and ICs that may have been tinkered with are correctly installed, not reversed, and with no shorted pins. Perform the peripheral card check, but don't leave the computer on with the clicking noise for any longer than necessary. The peripheral card check will indicate whether the motherboard or a peripheral card is the probable cause. It can be further determined that the problem is not in the keyboard or numeric keypad by disconnecting the keyboard plug and numeric keypad plug (if there is one) from their motherboard sockets.

If you were unable to isolate the exact problem cause, it may be time for you to take your system to a dealer for repair. At this point, you will be able to describe the symptoms to the service technician on a level which will be helpful to him. He will probably verify a short to ground exists with a multimeter and try to isolate the short by removing ICs from the motherboard, peripheral card, or auxiliary card in groups until the short goes away. You can do this yourself, but beware. Even experienced technicians damage ICs or install them incorrectly on occasion. If you remove all the ICs from a card and put them back in, you very well may create some casualties that weren't there when you started. Also, the problem may be more difficult than a dead short, and you might end up requiring a board replacement if it is too difficult.

The second bad power supply symptom that can be observed is a completely dead Apple with no clicking noise. This can be verified to be a power supply problem by measuring the +12, −12, +5, and −5 volt lines at any peripheral slot with a multimeter. If the voltages are good, then a timing generator problem is indicated. If incorrect voltages are present, there is probably a power supply problem which will require a swap out from a dealer. If no voltages are present, a dealer may still swap out the power supply, but you may only have a bad switch.

In the event the power supply switch fails, you have two options. You may take the computer to a dealer who might verify the switch is bad and replace it or who might insist on a power supply swap out. You may also verify the switch is bad yourself and replace it. Any Apple dealer will carry the switch or can obtain it from Apple. Here is a rough procedure for replacing the switch.

1. Turn the Apple off.
2. Remove the power cord from the power supply.
3. Disconnect the power supply plug from the motherboard connector.
4. Remove the power supply (four screws through Apple base plate).
5. Remove five screws from both sides of the power supply and separate top from bottom.
6. Verify switch casualty with ohmmeter.
7. Replace switch.
8. Reassemble and reinstall the power supply by reversing the dismantling procedure.

DO NOT APPLY POWER TO THE POWER SUPPLY WHILE INTERNAL COMPONENTS ARE EXPOSED. THE VOLTAGES INSIDE ARE VERY HAZARDOUS WHILE POWER IS APPLIED.

Peripheral Failures

It is sometimes fairly obvious that the problem lies only in a peripheral or its interface card. If everything else works, but a printer won't print, it's

pretty cut and dried. Other peripheral failures are less obvious. Cards that steal memory addressing such as a firmware or RAM card are so integrated into the overall operation that when one fails, symptoms can be the same as motherboard failures.

When you are certain that a fault lies with a peripheral, there are some steps you can take to try to determine the exact cause. First, with the computer off, remove all the other peripheral cards and the auxiliary card to be certain that one of them isn't somehow causing the problem. If that doesn't help, turn the computer off, and give the suspect peripheral card a visual inspection. Assuming the ICs are mounted in sockets, wiggle them all to make sure that they're properly seated. Verify that any plugs are properly installed. Clean the contacts of the card's edge connector with a pencil eraser or with alcohol and cotton swabs, preferably the latter. Reinstall the card and verify that the problem still exists. You can perform the same steps on any cards mounted in the peripheral itself.

Just the act of removing a peripheral card and installing it will often cure many problems, at least temporarily. Some lower quality contact materials will tend to make poor electrical contact when the temperature rises. Just wiggling the card can cure the problem, but be sure to wiggle it with the computer turned off. Cards with gold plated contacts are much less likely to cause this sort of trouble.

Another thing you can try is to run the peripheral in a different slot than its ordinary one, assuming it is not slot dependent. If it is slot dependent, try running another peripheral in that slot, the object being to prove there is nothing wrong with the slot's signals or connections. If the problem persists, you may as well start calling computer dealers to find one who will work on your peripheral.

If you cause a malfunction by removing a card with power applied, suspect that ICs connected to the INHIBIT' line are burned out. This includes the 74LS09 on a firmware card or a RAM card. You can burn up these LS09s by removing any card from any slot with the power on and aecidently shorting pin 32 (INHIBIT') to pin 33 (−12V). If you are less lucky, you might short pin 50 (+12V) to pin 49 (D0). In the latter instance, you may possibly destroy numerous ICs.

Here is one last tip for a special situation. I have known the LS125 on the Disk II analog card to be damaged on three separate occasions. On two of those occasions, the LS125 failure was caused by a person plugging the 20-pin ribbon cable connector into the controller incorrectly. The symptom was that the drive was always configured for writing even when the controlling program was attemptin to read data. Booting or cataloging a disk, for exam ple, would clobber the disk. A typical operator wil clobber several disks under these circumstance before realizing what he is doing. He thinks he i trying to read bad disks, but he is really makin disks go bad while trying to read them. If yo encounter these symptoms, try replacing the LS12 on the analog card of the offending drive or drives Then verify operation with disks containing non critical data. This tip also pertains to most second source 5-¼ inch drives available for the Apple. They generally use an LS125 for the same functions as the one in the Disk II drive.

Other Symptoms

There are no error lights built into the Apple, but there are some very distinct error indications which you can interpret if you understand the Apple. First, does it beep when you turn it on? The beep isn't made by some oscillator. It's made by programmed control of the speaker by the MPU. It means that a power-up RESET was generated, the 6502 works and is capable of address bus and data bus control and data bus reception, the E0—FF and C1—DF ROMs work, timing works, page 0 and 1 of RAM work, the MMU works, a good portion of the address decoding circuitry works, and the IOU is capable of toggling the SPKR line in response to address bus commands. In other words, if the speaker beeps, the Apple is in pretty fair shape.

The second big indicator of the nature of a problem is the video display. The scanning of RAM for video output is done independently of the MPU and the address bus. The only common bonds are timing and the data bus. The address bus, MPU, MMU, RAM, and C1—DF and E0—FF ROMs can all be burned to a crisp, and you will still have the display window on the screen. The contents would be jibberish, but the window would be there with its black margins on all sides. The presence of the window means that timing, the video scanner, and most of the video generator are working.

The four possible combinations of the two operational indicators can greatly narrow the field of possible causes of a given problem. The following interpretations of symptoms should help. In all cases perform the peripheral card check, visually inspect the motherboard, and verify all ICs are properly seated. Refer to Figure 10.3 (color section) to see which functional areas are affected by various ICs.

1. **No beep, no display.** This is one dead Apple and power supply problems are the probable cause. This can be verified by measuring the +5V, −5V, +12V, and −12V supplies at any peripheral slot with a multimeter. If any of the voltages is incorrect, read the **Power Supply Problems** section of this chapter for an indication as to how to proceed. If the power supply voltages are good, timing generator problems are indicated. Verify the presence of the timing generator signals using a logic probe or oscilloscope. If you have a second Apple IIe or simply have access to spare ICs, swap out the motherboard timing HAL, LS125, LS109, and IOU one by one to see if one of these ICs is causing the problem.
2. **No beep, display window present.** Timing is good, but the computer does not execute a stored sequential program. Many things could cause this symptom including the MPU, the IOU, the MMU, a ROM chip, a RAM chip, or any bad chip connected to the address or data bus. You may try to get an indication from the firmware diagnostics, but they probably will not run. Use a logic probe or oscilloscope to verify whether or not signals are normal at all pins, at one of the peripheral slots, and at the auxiliary slot. Pay particular attention to the address bus, the data bus, INHIBIT', RESET', and other 6502 control lines. The condition of the peripheral slot signals should guide you to possible causes.
3. **Beep, no display window.** Most of timing is good, but check LDPS' and VID7M with a logic probe or oscilloscope. Also check the IOU outputs related to video generation and the PICTURE' signal (see Figure 8.5). Verify that the firmware diagnostics operate correctly by holding open and close Apple while pressing CONTROL-RESET and listening to the speaker. Suspect the IOU if you don't hear the RAM test speaker tones. If you have access to spare ICs, you can replace the ICs and transistors pictured in Figure 8.5 one by one to see if one of them is causing the problem.
4. **Beep, window present, programs crash.** The computer executes programs but gets into trouble in certain cases. If they will operate, run the firmware diagnostics with all peripheral cards installed and again with peripheral cards removed. Follow up any leads the firmware diagnostics give you. If the diagnostics will not run, suspect RAM or ROM. If the diagnostics run and pass, recycle them for a long time using the procedure given in **The Firmware Diagnostics** section. Check all the signals on a peripheral slot with a logic probe or oscilloscope.

Intermittent problems are the bane of all computer servicemen as well as computer users. They are often dependent on the temperature and the product of a mechanical defect like imperfect electrical contact. They are sometimes impossible to repair in a cost effective way and the best thing that can be done in this instance is to swap out the defective assembly at a computer dealership. There are two tricks which can be used to make the problems become more regular so the causes can be identified. One is to subject the equipment to a severe mechanical jolt, like Humphrey Bogart in *The African Queen*. This will tend to prove or disprove the notion that there is a mechanical problem. Less drastically, you can flex the motherboard and reseat suspect peripheral cards and ICs. The second trick is to raise and lower temperature to make the problem occur. "Cold problems" can be made to appear by spraying cold spray, available at electronic stores, on suspected areas. "Hot problems" can be made to appear by directing a heat gun, or, less effectively, a hair dryer at the suspected area.

There are no doubt many symptoms not covered by the guidelines that have been given here. The intention was only to give some helpful hints, not a full blown maintenance aid. A serious reader of this book, however, should be in a very good position to correctly interpret the symptoms of any malfunctions which occur in his machine. In the absence of sophisticated diagnostic aids, full understanding of operation is the most important asset in isolating faults in a digital computer. It is good for Apple owners to possess this level of understanding, and it is also good for them to locate computer dealerships which employ service personnel with this level of understanding.

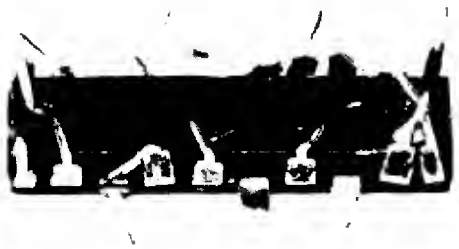


Figure 10.4 Some People Just Shouldn't Handle ICs.

glossary

address bus. A multi-line electrical connection from the MPU to various devices in a microcomputer by which the MPU specifies the location with which it will communicate. The address bus in the Apple IIe is made up of 16 lines, and the MPU can specify 65536 different addresses for data transfer. See data bus, R/W' line.

ampere, amp. The unit of measure of electrical current.

analog. Pertaining to quantities which vary through a continuous range such as a voltage which ranges from +5V to −5V. See digital.

AND gate. A logic gate from which the output will be true if and only if all of its inputs are true.

Applesoft BASIC. The floating point BASIC interpreter language written for the Apple II by Microsoft, Inc. and distributed by Apple Computer, Inc.

ASCII, American Standard Code for Information Interchange. A code for representing numbers, letters, and symbols in computers. ASCII is used in the Apple IIe for representing text in the keyboard input, text screen map, printer output, BASIC strings, and DOS text files.

assembler. A program which converts an assembly language source file into a machine language object file.

assembly language. A language which specifies machine language commands on a one to one basis but in which the computer manages many of the details of generating machine language code.

auxiliary slot. The 60-pin receptacle located on the left side of the Apple IIe motherboard. The auxiliary slot is designed to accept cards that support the Apple IIe 80-column video and 64K auxiliary RAM capabilities and functions related to video and timing generation.

bandwidth. A frequency range. Usually, the frequency range of sinewaves that will be processed or passed by an electronic circuit or signal path.

bank switching. A method of accessing more memory locations than the normal addressing range an MPU will allow. In bank switching, the MPU is allowed by hardware to address more than one memory bank using the same address range. An example is the $D000—$FFFF range of the Apple IIe which is bank switched between motherboard ROM, motherboard RAM, and auxiliary card RAM.

BASIC, Beginners All-purpose Symbolic Instruction Code. The primary high level language used in personal computers. It was originally developed at Dartmouth College as a training language, and has been developed into a powerful and usable tool by the microcomputer industry.

binary numbering system. A system based on powers of 2, as opposed to powers of 10 in the decimal system. The two symbols of the binary system are 0 and 1. See hexadecimal.

bit. A 2-state unit of information. The information is in one state or the other—on or off, for example.

bomb, crash. When a program is bombed or when it simply crashes, it loses control of the computer and must be restarted and possibly reloaded. It is particularly likely for a program to crash when it is first written and still has bugs in it.

bonding options. Optional characteristics of a custom IC that are determined when the IC is manufactured by bonding signal lines to low or high signal points. For example, selection of Apple IIe or Apple IIc operational characteristics is a bonding option of the MMU and IOU.

bootstrap. The process by which a computer loads large operating systems using a small firmware program.

buffer. (1) A temporary holding area in memory in which data resides before, during or after transfer operations. (2) Any hardware device which provides electrical isolation between two electrical points or sets of points.

bus. A multi-line electrical connection which distributes an associated group of signals among two or more communicating devices.

bus driver. A group of amplifiers which allow a group of signals to control a heavily loaded bus. The driver gives electronic leverage to the control signals so they can "drive" many devices.

byte. A group of 8 bits. The 6502 is an 8-bit MPU and therefore transfers and manipulates data one byte at a time.

BYTE FLAG. A term used in this book to describe the sync bit which leads groups of eight bits in Apple DOS data formats.

card cage. A row of receptacles into which printed circuit cards with edge connectors are plugged. The receptacles are wired in the back as is necessary to implement the functions of the cage.

cathode ray tube, CRT. A device in which a screen display is created by a high velocity stream of electrons striking a phosphor coating. The impact point on the screen is controlled by deflecting the electron stream via an electromagnetic or electrostatic field. The picture tube in a television is a CRT.

central processing unit, CPU. The electronic assembly which performs the arithmetic and logical operations of a computer.

chip. See integrated circuit.

COLOR BURST. In a television color video signal, a short sample of the COLOR REFERENCE signal which occurs just after the horizontal sync pulse. From the COLOR BURST, a television or monitor can reconstruct the COLOR REFERENCE.

compiler. A program which converts high level language source programs into machine language object programs.

complement. The complement of a binary number is a binary number in which binary "1"s replace "0"s, and "0"s replace "1"s in the original number. For example, 11010 is the complement of 00101.

complementary colors. Pertaining to the Apple IIe, colors produced by signals 180 degrees out of phase with each other—HIRES40 green and HIRES40 violet for example.

composite video. A complex video display signal containing horizontal and vertical sync, luminance and chrominance signals, and a color burst. The video output of the Apple IIe can loosely be called composite video.

CSW, Character output SWitch. See I/O links.

current. The motion of charged particles due to voltage. Generally, in electronics, the movement of electrons through conductive paths. Current is measured in amperes.

custom IC. An IC which is manufactured to suit the special needs of an equipment manufacturer. The IOU and MMU are examples of custom ICs.

cycle stealing DMA. Direct memory access from a device other than the MPU which occurs while MPU execution cycles are inhibited. Cycle stealing DMA slows execution of the MPU program.

data bus. A multi-line electrical connection over which data passes between the MPU and various devices in a microcomputer. The data bus in the Apple IIe is made up of 8 lines, so one byte can be transferred per MPU cycle. See address bus, R/W' line.

debounce. To eliminate signal variations resulting from the short period of unstable contact that occurs after a mechanical switch is thrown.

debug. To perfect a program by removing the bugs (defects) from it.

decimal numbering system. The system by which we normally represent numeric quantities. Ten symbols (0—9) represent quantities, while the position of each symbol in a number represents the significance or weight of that symbol. The weight increases by powers of ten as position shifts right to left.

digital. Pertaining to quantities which vary in discrete increments such as integer numbers. See analog.

DIIDD, Disk II Device Driver. The subroutine of ProDOS that is called to read or write a 512-byte block of floppy disk data. See RWTS.

DIP, Dual In line Package. A type of electronic component structure in which pins run lengthwise in two parallel rows. All ICs in the Apple IIe are DIP ICs, but there are some SIP (single in line package) resistor networks.

disassembler. A program which attempts to interpret data in memory as a machine language program and converts it to an assembly language listing. A firmware disassembler in the Apple IIe can be called via the monitor "L" command.

DMA, Direct Memory Access. Direct access to memory from devices other than the MPU. In the Apple IIe, data is directly accessed in memory by the video scanner/video generator combination without MPU participation. Additionally, a card in any peripheral slot can directly access memory and other mother board devices by pulling the DMA' line low.

dot matrix. A method of forming displayed or printed characters in which individual dots at fixed positions

in a matrix are displayed as necessary to form the characters.

DOUBLE-RES mode. A term used in this book to refer to the double horizontal resolution (motherboard and auxiliary card map) display mode of the Apple IIe. DOUBLE-RES mode is further categorized as the TEXT80, LORES80, and HIRES80 modes. See SINGLE-RES mode.

driver. (1) A program which manages specific hardware or I/O functions, e.g. a printer driver. (2) A hardware device that supplies signals at the voltage and current required by other hardware devices, e.g. the Apple IIe bidirectional peripheral data bus driver.

Dvorak keyboard layout. An alphabetic keyboard layout that is designed to allow faster touch typing speed than the standard QWERTY layout. Dvorak is available as an alternate layout in the keyboard ROM of the American Apple IIe.

dynamic memory. Memory in which data will bleed off and lose its validity if it is not regularly refreshed. RAM in the Apple IIe is dynamic and must be refreshed every 2 milliseconds.

Easter egging. A troubleshooting method where possibly failed components or assemblies are replaced with known good units. Easter eggers can sometimes repair equipment they know little or nothing about.

edge sensitive input. An electronic circuit input which is sensitive only to changes in signal level. The 6502 NMI' input is an edge sensitive input which responds only to negative voltage transitions. See level sensitive input.

EPROM, Erasable Programmable Read Only Memory. A type of PROM that can be erased and reprogrammed. Subtypes include UVEPROM (ultraviolet light erasable PROM) and EEPROM (electrically erasable PROM). See ROM, PROM.

exclusive OR gate. A logic gate from which the output will be true if and only if at least one, but not all, inputs are true.

firmware. Programs and data stored in ROM. Firmware determines many of the operational features of the Apple IIe.

flag. (1) A memory location used by a program to signify some sort of status. A common way to use a location as a flag is to set or reset its most significant bit. (2) A readable indicator of hardware status such as the status flags of the 6502, MMU, and IOU.

flip-flop. A 1-bit storage device capable of storing data in response to its logical inputs and a clockpulse. Registers of older computers were comprised of a number of flip-flops with a substantial amount of associated logic gating.

float. If all devices capable of controlling the voltage on an electrical conductor are isolated from the conductor, the conductor is said to float. In the Apple IIe, all conductors on the address bus or data bus can be isolated from control, so these buses sometimes float. Logic which creates this condition floats the bus.

flux. Lines of force used to represent a magnetic field. The lines of force provide a mental picture for visualizing the substance of a magnetic field. In theoretical calculations, field strength is proportional to flux density.

font, character. Patterns of ONEs and ZEROs stored in memory which represent the dot image of dot matrix text or graphics characters.

frequency response. The response of an electronic circuit or device as frequency varies. Good high frequency response is required in video monitors used with high resolution video computers like the Apple IIe.

gate. A logic circuit having one output and more than one input. Like a gate in a fence, the logic gate allows intelligence to pass when the inputs are correct. AND gates and OR gates are two types of gates. When an input activates a logic device, it is said to "gate" it on.

general purpose computer. A computer whose stored program may be altered to change its purpose. This is normally achieved by storing the program in random access, read/write memory. See special purpose computer.

hacker, hack, computer hack. A person who builds or modifies computer electronic assemblies, mostly for fun. More loosely defined, any person who thinks computers are fun.

HAL, Hard Array Logic. An IC whose logical functions, within a given framework, are fixed when it is manufactured. HAL is similar to masked ROM except that logical functions, instead of stored data, are specified. See PAL.

handler. A program designed to handle a specific occurrence such as an interrupt or system reset.

hardware. The components and assemblies of which a computer and its peripherals are made.

Hertz, Hz. A unit of frequency measurement which used to be referred to more sensibly as cycles per second or cps.

hexadecimal numbering system. A system based on powers of 16, as opposed to the powers of 10 in the decimal system. The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

high level language. A computer language whose commands correspond to machine language routines. High level languages are easy to use and powerful. The predominate high level language of the Apple IIe is Applesoft BASIC.

high RAM. A term used in this book to refer to RAM in the Apple IIe that is addressed at $D000—$FFFF.

horizontal scan. The movement of the electron beam in a television from left to right across the face of the CRT.

impedance. The quality of hindrance of an electrical device to current flow at a given signal frequency or range of signal frequencies. Impedance, like resistance, is measured in ohms. In computer 3-state logic, the three states are high voltage, low voltage, and high impedance (high isolation).

I/O links, I/O hooks. Memory locations which contain the addresses of the primary input and output routines of the Apple IIe. The output link is CSW (Character output SWitch, locations $36 and $37), and the input link is KSW (Keyboard input SWitch, locations $38 and $39).

I/O port. The conceptual entry point through which data flows in I/O operations. For example, in the Apple IIe, $C000 is the address of the keyboard input port, and keyboard data is loaded from address $C000.

input/output, I/O. The process of moving information to and from a computer, as in keyboard input, video output, disk I/O, and printer output.

Integer BASIC. The original BASIC interpretive language, supplied with the Apple II and Apple IIe. It was written by Steve Wozniak, the principle designer of the original Apple II computer. Integer BASIC is included on the DOS 3.3 system master and can be loaded to and run resident in Apple IIe high RAM.

integrated circuit, IC, chip. An electronic component into which the functions of many other components are integrated. Typically, a chip will be the equivalent of thousands of diodes, transistors, and resistors.

interface. Communication circuitry between two devices, such as the interface between an Apple IIe and a printer.

interlacing. A technique in which alternating television vertical scans are displaced from each other. This increases vertical resolution without introduction of screen flicker.

interpreter. A program which interprets stored sequences of high level commands and executes them via fixed machine language subroutines. The Apple IIe is supplied with Applesoft BASIC interpreter in ROM.

interrupt. A signal or instruction which, when active, causes a computer to interrupt sequential program execution and branch to an interrupt handling program. The 6502 has four types of interrupts: RESET, the Non-Maskable Interrupt, the Interrupt Request, and the BREAK instruction.

IOU, Input/Output Unit. A custom IC in the Apple IIe and Apple IIc which performs logical functions related to video output and other I/O.

joystick. A device which converts the two dimensional motion of a lever into measurable electrical equivalents of the X and Y components of the motion. Normally an Apple IIe joystick will be made of two potentiometers, one which responds to y-axis motion of the lever and one which responds to x-axis motion.

KSW, Keyboard input SWitch. See I/O links.

least significant bit, LSB. The bit of a binary word or number which has the least weight or significance. The rightmost bit of a binary number.

level sensitive input. An electronic circuit input that responds to stable voltage levels, not just changes in voltage levels. The 6502 IRQ' input is an example of a level sensitive input. See edge sensitive input.

linear IC. A type of IC used in linear amplification and other analog functions, as opposed to digital switching functions. Some linear ICs used in the Apple IIe are 556 (on the disk controller) and 558 timers and the 741 cassette input amplifier.

LSTTL, Low powered Schottky TTL. A type of TTL which provides a good compromise of high speed and low power consumption. Most Apple IIe TTL is LSTTL. The name comes from Schottky-Barrier clamping and coupling diodes inside the IC.

machine cycle. A clocked cycle of an MPU. The machine cycle of the 6502 in the Apple IIe is the period between high to low transitions of the PHASE 2 clock.

machine language. The language of the central processor of a computer (6502 machine language in the Apple IIe).

mainframe computer. When computers were physically large, the structure which held the central processor was called the mainframe. Computers which have such separate structures are mainframe computers.

Megahertz, MHz. One million Hertz. One million cycles per second.

memory cell. A portion of memory capable of storing one bit of information.

memory mapped I/O. A method of I/O implementation in which addresses are assigned to I/O functions. The Apple IIe uses memory mapped I/O, and Apple IIe I/O device addresses are in the $C000—$CFFF range.

memory mapped video. A method of computer video display generation in which a map of the screen display is placed in memory. In the Apple IIe, the MPU builds the screen map in memory, and the memory map is independently scanned for video processing by video scanning circuitry in the IOU.

microprocessing unit, MPU, microprocessor. The 1-chip central processing unit of a microcomputer. This definition is a good subject for an argument.

microsecond, μsec. One millionth of a second. One thousand nanoseconds.

millisecond, msec. One thousandth of a second. One thousand microseconds.

MMU, Memory Management Unit. A custom IC that manages overall response to MPU addressing in the Apple IIe and Apple IIc.

mnemonic. That which is intended to assist our memories. MPU op codes reside in memory as binary numbers, but they are referred to by mnemonic labels such as LDA (LoaD Accumulator) in assembly language programs.

modulate. To vary a high frequeney signal as a function of a lower frequency signal. The video output signal of the Apple IIe can be used to amplitude modulate a television frequency signal, and that modulated signal can be received by a television set. The high frequency signal carries the video to the TV and is called an RF carrier.

monitor. A program which provides for communication with a computer at a very basic level. Common capabilities include program start up, memory modification, and monitoring of computer registers.

monitor, video. An electronic device which generates a display from a video input signal. It is not capable of television radio frequency signal reception.

monochrome. Of one color. A name for black and white television which accurately describes the fact that there is only one color tone displayed.

MOS integrated circuit. A chip using metal-oxide semiconductor technology. MOS ICs in the Apple IIe include the 6502, ROM, RAM, the IOU, and the MMU.

most significant bit, MSB. The bit of a binary word or number which has the greatest weight. The leftmost bit of a binary number.

motherboard. A printed circuit card into which smaller printed circuit cards can be plugged.

multimeter. An instrument which can measure electrical voltage, resistance, or current.

multiplex. To combine multiple sources of information onto one line. There are a number examples of time-multiplexing in the Apple IIe in which several possible signals are switched onto a line, one after the other.

NAND gate. A logic gate from which the output will be false if, and only if, all of its inputs are true.

nanosecond, nsec. One billionth of a second.

negative logic. A system of logic analysis in which the high voltage state is considered to be false or zero, and the low voltage state is considered to be true or one. See positive logic.

NOR gate. A logic gate from which the output will be false if, and only if, any of its inputs are true.

NTSC, National Television Systems Committee. The television system used in the United States and some other countries. See PAL, SECAM.

NTSC motherboard. The version of the Apple IIe motherboard that is operated in the United States and other countries that use the NTSC standard television system. See PAL motherboard.

object file. The result of an operation which processes a group of data and stores the result elsewhere. In assembly language processing, the assembly language source file is assembled into a machine language object file.

octal numbering system. A system based on powers of 8, as opposed to the powers of 10 in the decimal system. The eight symbols of the octal system are 0, 1, 2, 3, 4, 5, 6, and 7.

ohm. The unit for measuring electrical resistance and impedance.

ohmmeter. A device which measures electrical resistance. Usually resistance is measured with a VOM (Volt-Ohm Meter) or multimeter which performs other functions besides resistance measurement.

op code, operation code. The part of a machine language instruction which specifies the command which is to be performed. The op code of each 6502 instruction is the first byte of that instruction.

open collector. A class of TTL IC outputs which are tied only to an open transistor collector inside the IC. Open collector outputs are useful for driving wire-OR lines and interfacing to non-TTL devices.

operand. The entity operated on by a machine language instruction. In "LDA $00", the operand is the contents of memory location 0. In "SEC", the operand is the carry bit of the 6502 Status Register.

OR gate. A logic gate from which the output will be true if, and only if, any of its inputs are true.

oscillator. An electronic circuit that generates an output signal that alternates between high and low voltage peaks. The 14M signal is produced by a 14.31818 MHz oscillator in the 60 Hz Apple IIe.

oscilloscope. A test instrument which produces a cathode ray tube display of a test voltage, plotted against time.

page. (1) The 6502 memory addressing range of $10000 bytes is divided up into $100 pages of $100 bytes each. $000—$0FF is Page 0; $100—$1FF is Page 1; etc. (2) There are four RAM address ranges that can be scanned for video output in the Apple IIe. These are TEXT/ LORES PAGE 1, TEXT/LORES PAGE 2, HIRES PAGE 1, and HIRES PAGE 2. PAGE 1/PAGE 2 selection in the Apple IIe is performed via programmed manipulation of the PAGE2, 80STORE, and HIRES soft switches.

PAL, Phase Alternating Line. The television system used in most western European countries and some other countries. See NSTC, SECAM.

PAL, Programmable Array Logic. An IC whose logical functions, within a given framework, can be programmed after the IC is manufactured. PAL is similar to PROM except that logical functions, instead of stored data, are programmed. See HAL.

PAL motherboard. The version of the Apple IIe motherboard which is operated in countries that do not use the NTSC television system. PAL motherboards have a PAL color video encoder built in. See NTSC motherboard.

parallel data transfer. Simultaneous transfer of n bits of data on n lines, as in 8-bit parallel data transfer between the MPU and memory in the Apple IIe.

peripheral slots, peripheral bus, Apple bus. The seven slots in the back of the Apple IIe and their associated electrical connections.

phase. The angular position of a cyclic event referenced to some event of the same frequency. For example, HIRES violet video is 180 degrees out of phase with HIRES green video.

pipelining. A process by which program execution speed is increased in the 6502. In pipelining, the next instruction's op code is fetched during the last execution cycle of instructions which do not write to the data bus.

poll. In programming, to repeatedly examine an addressed location or to examine a series of addressed locations until a certain indication is given, e.g. polling the disk controller data register for an MSB set indication.

positive logic. A system of logical analysis in which the high voltage state is considered to be true or one, and the low voltage state is considered to be false or zero. The Apple IIe and most modern computers use positive logic. See negative logic.

potentiometer, pot. A mechanically variable resistor. Typically, resistance will be proportional to the shaft rotation of the pot. Apple IIe paddles are pots with knobs on them.

power supply. An electronic assembly which converts an AC (alternating current) line voltage to usable DC (direct current) power. The Apple IIe power supply converts household power to +12V, −12V, +5V, and −5V referenced to ground.

printed circuit card, PC board. A thin card made of an insulating material upon which electronic components are mounted. The component wiring is "printed" on the board. The printing process involves starting with a card completely coated with conductive metal, then etching away everything but the desired conductive parts using photo-chemical methods.

program counter. A counter in a computer which contains the memory address of the instruction being executed. The 6502 has an internal 16-bit program counter.

PROM, Programmable Read Only Memory. Read only memory that can be programmed after it is manufactured. See ROM, EPROM.

propagation delay. The time it takes for a signal or voltage to travel between two points. In logic gating, the time required for an output to respond to a change in associated inputs.

R/W' line, read/write line. The signal by which the MPU indicates whether it will receive data from or transfer data to an addressed device. The R/W' line is best thought of as an extension of the address bus. See address bus, data bus.

RAM, alterable memory, read/write memory. Memory in which a computer can store or access data.

random access memory. Memory in which any location can be accessed at will, such as the RAM and ROM in the Apple IIe. See serial access memory.

raster. The pattern of scan lines produced on the screen of a monitor or television. The 60 Hz Apple IIe raster contains 262 lines with no interlacing.

read cycle. A machine cycle in which the MPU receives data from the addressed location via the data bus.

read-modify-write instructions. A class of MPU instructions in which an operand is read from a memory location, then modified, then returned to the same memory location. 6502 read-modify-write instructions include ASL, LSR, ROL, ROR, INC, and DEC.

refresh. (1) The process of renewing data in dynamic memory before it bleeds off. (2) The process of renewing an image on a cathode ray tube by rescanning the image before it fades away.

register. A temporary storage device which holds more than one bit of information. It will normally serve some special logic purpose. Examples include the 6502 internal registers and the data register of the Disk II controller.

relocatable program. A program which does not have to be in one specific memory range to be properly executed. It can be relocated to different memory areas for execution.

resistance. The quality of hindrance of an electrical device to direct current flow. Resistance in a current path may be controlled by installing fixed or variable resistors. The unit of measurement of electrical resistance is the ohm.

retrace. In electron beam scanning across the face of a television or monitor picture tube, the very fast return of the beam from right to left during which the picture is blanked. See trace.

RF modulator. A device which varies a high frequency signal as a function of a lower frequency signal. See modulate.

ROM, read only memory, non-volatile memory, non-alterable memory. A type of memory which the computer cannot write to or otherwise alter. It holds programs and data which are always available when power is applied, such as BASIC and the monitor in the Apple IIe. See PROM, EPROM.

RWTS, Read or Write a Track and Sector subroutine. The subroutine of DOS 3.2 and DOS 3.3 that is called to read or write the data field of a floppy disk sector or to format a floppy disk. See DIIDD.

SECAM, SEquential Color And Memory. The television system used in France and some other countries. There are at least two versions of SECAM in use. See NSTC, PAL.

serial access memory. Memory which can only be accessed by sequencing through locations until the correct location is found. High speed magnetic tape and bubble memory are examples of serial memories. See random access memory.

serial data transfer. Transferring data one bit at a time over a single line, as in the shifting of text patterns to the PICTURE signal.

serrations. Narrow horizontal sync pulses superimposed on the long vertical sync pulse in a video signal that prevent televisions and monitors from becoming horizontally unstable during the vertical sync pulse. See Figure 8.2.

simultaneous DMA. Direct memory access from a device other than the MPU which occurs when the MPU is not communicating with memory and which does not slow execution of MPU programs. Video scanner access to RAM is an example of simultaneous DMA in the Apple IIe.

SINGLE-RES mode. A term used in this book to refer to the normal horizontal resolution (motherboard map only) display mode of the Apple IIe. SINGLE-RES mode is further categorized as the TEXT40, LORES40, and HIRES40 modes. See DOUBLE-RES mode.

SIP, Single In line Package. A type of electronic component structure in which pins run lengthwise in a single row. There are several SIP resistor networks in the Apple IIe.

soft switch. A conceptual switch that can be turned on or off by programmed references to its on or off address. Soft switches are used for a multitude of control functions in the Apple IIe including memory management and display mode control.

software. Programs and data stored in RAM and on storage media such as disks.

source file. A source of data for data processing. In assembly language processing, the assembly language source file is assembled into a machine language object file.

special purpose computer. A computer whose function cannot be changed by altering a stored program. The functions of a special purpose computer may be hard wired, or the computer may execute fixed programs stored in ROM.

stack. In microcomputers, an area of memory set aside for temporary storage and subroutine return link information. In programming, the stack is conceptually similar to a stack of cards which can be drawn from or discarded to, one card at a time.

static memory. Memory which requires no refreshing to retain its data, such as the static ROM on the Apple IIe motherboard or the static RAM on the 1K auxiliary RAM card.

status register. A register in a central processor which contains control information. In the 6502, the status register contains indicators of the logical results of various executed commands.

strobe. A short pulse that performs a triggering or clocking action. Strobes in the Apple IIe include RAS', CAS', the C040 STROBE', and the keypress strobe from the keyboard encoder.

SYNC (6502). A 6502 output signal that goes high when the 6502 is fetching an op code. The 6502 SYNC signal is connected to pin 39 of the seven Apple IIe peripheral slots.

television sync. That part of the television signal which synchronizes the scanning of the electron beam in the CRT. It includes horizontal and vertical sync.

trace. In electron beam scanning across the face of a television or monitor picture tube, the scan from left to right during which the the picture is displayed. See retrace.

tri-state logic, three-state logic. A logic system in which there are three states: high voltage, low voltage, and high impedance. Devices connected to the Apple IIe data bus have tri-state outputs so the various devices are able to share control of the data bus.

troubleshoot. To isolate and repair the casualties in a failed piece of hardware.

TTL, Transistor-Transistor Logic. The logic family to which most general logic ICs in the Apple IIe belong. Both the inputs and outputs of TTL chips are connected to transistors inside the chip. As opposed to MOS devices like RAM, ROM, and the 6502, TTL circuits are made using bipolar technology.

underware. A substitute for firmware used by some of Apple's competitors to cut costs.

vector. An address or jump instruction stored in a memory location which contains program flow information in case of certain events. An example is the interrupt vectors stored in high memory in a 6502 based computer.

vertical scan. The movement of the electron beam in a television down the face of the CRT. In the 60 Hz Apple IIe, 262 horizontal scans occur during every vertical scan.

video. A signal which can be used to control the energy of the electron beam of a CRT, thus controlling display intensity, as in television video, radar video, and oscilloscope video. In television processing, the combination of picture, sync, and color information is commonly referred to as video.

video scanner, video scan counter. Terminology used by this book to describe a counter inside the IOU that controls memory scanning for video output in addition to controlling the television or monitor display scan.

volt. A unit for measuring voltage. Voltages of +12, −12, +5, and −5 volts are distributed throughout the Apple IIe.

voltage, electromotive force, EMF. A force of nature which, when present, causes charged particles to move; the force which causes electric current. Voltage is measured in volts.

wetware. A gray matter found within the craniums of most humans.

wire-OR, collector-OR. A low level OR gate formed by wiring various signals together. The RESET', IRQ', NMI', INHIBIT', and DMA' signals of the Apple IIe are examples of wire-OR connections. Any peripheral card may bring any of the wire-OR lines low, but cards not bringing a line low must present a high impedance to that line. If no peripheral card is bringing a wire-OR line low, a 3300 ohm motherboard resistor will pull the line high.

write cycle. A machine cycle in which the MPU sends data to the addressed location via the data bus.

appendix A

References

Apple Computer, Inc. *Apple II Reference Manual.* Jan. 1978.

Apple Computer, Inc. *Applesoft II BASIC Programming Reference Manual.* 1978.

Apple Computer, Inc. (Christopher Espinoza). *Apple II Reference Manual.* 1979.

Apple Computer, Inc. (originally by Phyllis Cole and Brian Howard). *The DOS Manual.* 1980.

Apple Computer, Inc. (Allen Watson). *Apple II Extended 80-Column Text Card Supplement for IIe Only.* 1982.

Apple Computer, Inc. (Allen Watson). *Apple IIe Reference Manual.* (preliminary). May 1982.

Apple Computer, Inc. (Allen Watson). *Apple II Reference Manual for IIe Only.* 1982.

Apple Computer, Inc. *DOS Programmer's Manual for II, II+, IIe.* 1982.

Apple Computer, Inc. *Reference Manual Addendum: Monitor ROM Listings for IIe Only.* 1982.

Apple Computer, Inc. *BASIC Programming With ProDOS.* 1983.

Apple Computer, Inc. *DOS User's Manual for II, II+, IIe.* 1983.

Apple Computer, Inc. *ProDOS Technical Reference Manual (for the Apple II family).* 1983.

Apple Computer, Inc. *Apple IIc Reference Manual.* Vol. 1 (preliminary). March 1984.

Apple Computer, Inc. *About Your Enhanced Apple IIe: Programmer's Guide* (preliminary). Sept. 1984.

Apple Computer PCS Division (John MacDougall). *How to Get at the DVORAK (American Simplified) Keyboard in the Apple IIe.* no date.

Bishop, Bob. "Have an Apple Split." *SOFTALK,* p. 54, Oct. 1982.

Brown, Chris and Maloney, Eric. "FCC Takes Aim Against RFI Polluters." *Microcomputing,* p. 30, April 1981.

Caffrey, Morgan P. "The New Apple IIe." *Apple Orchard,* p. 13, Feb. 1983.

Ciotti, Paul. "Revenge of the Nerds." *California,* p. 73, July 1982.

Derfler, Frank J. "Trying to Live in Harmony with Harmonics." *Microcomputing,* p. 36, April 1981.

Devry Institute of Technology. *Electronics Technology.* Volumes 7 and 10, 1971. This textbook was the primary source of information relating to NTSC television.

Fischer, Dan and Caffrey, Morgan P. "Go On and Interrupt Your Apple." *SOFTALK,* p. 47, March 1982 and p. 65, April 1982.

Fretwell, Cecil. "Setting I/O Hooks in ProDOS." *Call-A.P.P.L.E.,* p. 39, April 1984.

Gayler, Winston D. *The Apple II Circuit Description.* Howard W. Sams & Co, Inc. 1983.

Lancaster, Don. *TTL Cookbook.* Howard W. Sams & Co, Inc. 1974.

Leventhal, Lance A. *6502 Assembly Language Programming.* OSBORNE/McGraw-Hill, 1979.

Mazur, Jeffrey. "HARDTALK." *SOFTALK*, p. 208, Sept. 1982. Concerns disk drives.

Moore, Robin. "Apple's Enhanced Computer, the Apple IIe." *BYTE*, p. 68, Feb. 1983.

MOS Technology, Inc. *MCS6500 Microcomputer Family Hardware Manual.* 1976.

Osborne, Adam. *An Introduction to Microcomputers, Volume 1, Basic Concepts.* Adam Osborne and Associates, 1976.

Osborne, Adam and Kane, Gerry. *Osborne 4 & 8 Bit Microprocessor Handbook.* OSBORNE/McGraw-Hill, 1981.

Sather, Jim. *Understanding the Apple II.* Quality Software. 1983.

Sims, H. V. *Principles of PAL Colour Television and Related Systems.* Newnes-Butterworths. 1969.

Sippl, Charles J. *Microcomputer Dictionary,* 2nd edition. Howard W. Sams & Co, Inc. 1981.

Synertek, Inc. *SY6500/MCS6500 Microcomputer Family Programming Manual.* Aug. 1976.

Synertek, Inc. *Applications Information AN2, SY6500 Microprocessor Family, Microprocessor Products.* Oct. 1981.

Tommervik, Al with Hunter, David and Durkee, David. "The Apple IIe: The Difference." *SOFTALK*, p. 118, Feb. 1983.

Texas Instruments, Inc. *Designing with TTL Integrated Circuits.* 1971.

Watson, Allen III. "True Sixteen-Color Hi-Res." *Apple Orchard*, p. 27, Jan. 1984.

White, Robert M. "Disk-Storage Technology." *Scientific American*, p. 112, Aug. 1980.

Williams, Richard. "How to Use the Hooks." *MICRO*, p. 30:7, Nov. 1980.

Worth, Don and Lechner, Pieter. *Beneath Apple DOS.* Quality Software. 1981.

Worth, Don and Lechner, Pieter. *Beneath Apple ProDOS.* Quality Software. 1984.

Wozniak, Stephen. "The Apple II: System Description." *BYTE*, p. 34, May 1977.

Wozniak, Stephen. "SWEET16: The 6502 Dream Machine." *BYTE*, p. 150, Nov. 1977.

Wozniak, Stephen. Speech made at Applefest. Anaheim, CA, April 17, 1983.

Component Data:

Fairchild. *MOS Memory Data Book.* 1981.

General Instrument Corp. *Microelectronics Data Catalog.* 1982.

General Instrument Corp. *ROM 3.* no date.

Hitachi America, Ltd. *IC Memories.* no date.

Jameco Electronics. *1984 Catalog,* p 14. 1984.

MOS Technology, Inc. *6500 Microprocessors.* March 1980.

Monolithic Memories. *BIPOLAR LSI 1984 Databook.* Fifth edition. 1978, 1981, 1983.

Motorola Semiconductors. *MC3470 Floppy Disk Read Amplifier System.* 1978.

National Semiconductor. *Interface Databook.* 1980.

National Semiconductor. *Memory Databook.* 1980.

National Semiconductor. *Logic Databook.* 1981.

National Semiconductor. *Linear Databook.* 1982.

NCR Corporation. *NCR65C02.* 1982.

Rockwell International. *R6500 Microcomputer System Data Sheet.* Rev. 4, Nov. 1981.

Rockwell International. *R65C02, R65C102, and R65C112 R65C00 Microprocessors (CPU).* Document No 29651N52. Rev 2, Feb. 1984.

Synertek. *1981—1982 Data Catalog.* 1982.

Synertek. *1983 Data Book.* 1982.

Texas Instruments, Inc. *The TTL Data Book for Design Engineers.* 1976.

United Technical Publications. *IC MASTER.* 1981.

appendix B

Trademarks

The following is a list of Registered Trademarks referred to in the text of *Understanding the Apple IIe*.

Apple	Apple Computer, Inc.
Apple II	Apple Computer, Inc.
Apple II Plus	Apple Computer, Inc.
Apple IIc	Apple Computer, Inc.
Apple IIe	Apple Computer, Inc.
Applesoft	Apple Computer, Inc.
CP/M	Digital Research, Inc.
Donkey Kong	Nintendo
HAL	Monolithic Memories, Inc.
PAL	Monolithic Memories, Inc.
Microsoft	Microsoft, Inc.
Softcard	Microsoft, Inc.
TRI-STATE	National Semiconductor Corporation
Z80	Zilog, Inc.

appendix C

6502/65C02 Data

In attempting to analyze Apple timing, it is discouraging to find that the three 6502 manufacturers have different specifications, even though the MSC6502, R6502, and SY6502 should all perform identically. It is even more discouraging to find that the specifications are well beyond the range of typical operations. I therefore feel that Figure 4.5, which shows some measurements made of a Synertek 6502 in an Apple IIe, is a more realistic indicator of 6502 timing than the manufacturers' data sheets. Nevertheless, a partial reproduction of Rockwell International's data sheet is given here. This data is reprinted with the permission of Rockwell International Corporation, copyright 1981, all rights reserved.

Rockwell's timing specification charts are followed by the author's compilation of the 2 MHz timing charts of the Synertek, Rockwell International, and MOS Technology 6502 data sheets (Table C.1). Every attempt was made to make the data in this compilation faithfully represent the data contained in each manufacturer's data sheet. Table C.1 is followed by a layout of the author's which shows the execution periods of the various 6502 instructions (Table C.2).

Following the 6502 data is a partial reproduction of the NCR 65C02 data sheet. This data is reprinted with the permission of NCR Corporation, copyright 1982, all rights reserved. The 65C02 data sheet is included here because a growing number of owners will be using a 65C02 in their Apple IIe's now that Apple has released the firmware upgrade. Following the 65C02 data sheet is a layout of the author's which shows the execution periods of the various 65C02 instructions (Table C.3).

Rockwell

R6500 Microcomputer System DATA SHEET

R6500 MICROPROCESSORS (CPU)

SYSTEM ABSTRACT

The 8-bit R6500 microcomputer system is produced with N-Channel, Silicon Gate technology. Its performance speeds are enhanced by advanced system architecture. This innovative architecture results in smaller chips – the semiconductor threshold is cost-effectivity. System cost-effectivity is further enhanced by providing a family of 10 software-compatible microprocessor (CPU) devices, described in this document. Rockwell also provides memory and microcomputer system– as well as low-cost design aids and documentation.

R6500 MICROPROCESSOR (CPU) CONCEPT

Ten CPU devices are available. All are software-compatible. They provide options of addressable memory, interrupt input, on-chip clock oscillators and drivers. All are bus-compatible with earlier generation microprocessors like the M6800 devices.

The family includes six microprocessors with on-board clock oscillators and drivers and four microprocessors driven by external clocks. The on-chip clock versions are aimed at high performance, low cost applications where single phase inputs, crystal or RC inputs provide the time base. The external clock versions are geared for multiprocessor system applications where maximum timing control is mandatory. All R6500 microprocessors are also available in a variety of packaging (ceramic and plastic), operating frequency (1 MHz, 2 MHz and 3 MHz) and temperature (commercial and industrial) versions.

MEMBERS OF THE R6500 MICROPROCESSOR (CPU) FAMILY

Microprocessors with Internal Two Phase Clock Generator

Model	Addressable Memory
R6502	64K Bytes
R6503	4K Bytes
R6504	8K Bytes
R6505	4K Bytes
R6506	4K Bytes
R6507	8K Bytes

Microprocessors with External Two Phase Clock Input

Model	Addressable Memory
R6512	64K Bytes
R6513	4K Bytes
R6514	8K Bytes
R6515	4K Bytes

FEATURES

- Single +5V supply
- N channel, silicon gate, depletion load technology
- Eight bit parallel processing
- 56 Instructions
- Decimal and binary arithmetic
- Thirteen addressing modes
- True indexing capability
- Programmable stack pointer
- Variable length stack
- Interrupt capability
- Non-maskable interrupt
- Use with any type of speed memory
- 8-bit Bidirectional Data Bus
- Addressable memory range of up to 64K bytes
- "Ready" input
- Direct Memory Access capability
- Bus compatible with M6800
- 1 MHz, 2 MHz, and 3 MHz versions
- Choice of external or on-chip clocks
- On-the-chip clock options
  - External single clock input
  - Crystal time base input
- Commercial and industrial temperature versions
- Pipeline architecture

Ordering Information

Order Number: R65XX___

- Temperature Range:
  - No suffix = 0°C to +70°C
  - E = -40°C to +85°C (Industrial)
- Package: C = Ceramic, P = Plastic
- Frequency Range:
  - No suffix = 1 MHz
  - A = 2 MHz
  - B = 3 MHz
- Model Designator:
  - XX = 02, 03, 04, . . . 15

SPECIFICATIONS

Maximum Ratings

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	-0.3 to +7.0	Vdc
Input Voltage	V_{in}	-0.3 to +7.0	Vdc
Operating Temperature	T_A		°C
Commercial		0 to +70	
Industrial		-40 to +85	
Storage Temperature	T_{STG}	-55 to +150	°C

NOTE
This device contains input protection against damage to high static voltages or electric fields; however, precautions should be taken to avoid application of voltages higher than the maximum rating.

Electrical Characteristics

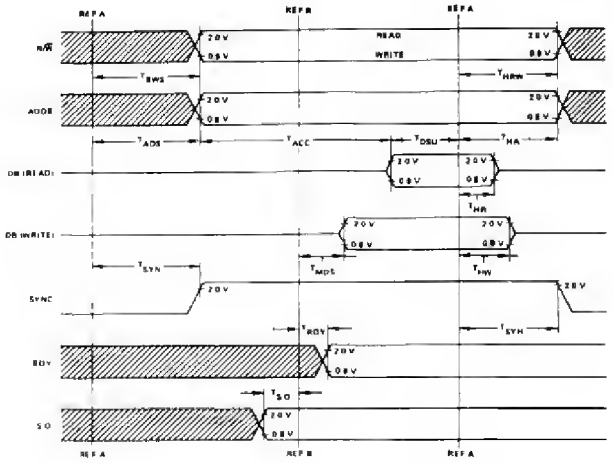
(V_{CC} = 5.0 ±5%, V_{SS} = 0)

ϕ_1, ϕ_2 applies to R6512, 13, 14, 15, $\phi_{o(in)}$ applies to R6502, 03, 04, 05, 06 and 07.

Characteristic	Symbol	Min	Max	Unit
Input High Voltage	V_{IH}			Vdc
Logic, $\phi_{o(in)}$		2.0	V_{CC}	
ϕ_1, ϕ_2		-0.3	V_{CC} + 0.25	
Input Low Voltage	V_{IL}			Vdc
Logic, $\phi_{o(in)}$		-0.3	0.8	
ϕ_1, ϕ_2		-0.3	0.4	
Input Leakage Current	I_{in}			μA
(V_{in} = 0 to 5.25V, V_{CC} = 0)				
Logic (Excl. Rdy, S.O.)		–	2.5	
ϕ_1, ϕ_2		–	100	
$\phi_{o(in)}$		–	10.0	
Three-State (Off State) Input Current	I_{TSI}			μA
(V_{in} = 0.4 to 2.4V, V_{CC} = 5.25V)				
Data Lines		–	10	
Output High Voltage	V_{OH}			Vdc
(I_{LOAD} = -100 μAdc, V_{CC} = 4.75V)				
SYNC, Data, A0-A15, R/$\overline{W}$, ϕ_1, ϕ_2		V_{SS} + 24	–	
Output Low Voltage	V_{OL}			Vdc
(I_{LOAD} = 1.6 mAdc, V_{CC} = 4.75V)				
SYNC, Data, A0-A15, R/$\overline{W}$, ϕ_1, ϕ_2			V_{SS} + 0.4	
Power Dissipation	P_D			mW
1 and 2 MHz		–	700	
3 MHz		–	800	
Capacitance at 25°C	C			pF
(V_{in} = 0, f = 1 MHz)				
Logic	C_{in}	–	10	
Data		–	15	
A0-A15, R/W, SYNC	C_{out}	–	12	
$\phi_{o(in)}$	$C\phi_{o(in)}$	–	15	
ϕ_1	$C\phi_1$	–	50	
ϕ_2	$C\phi_2$	–	80	

NOTE
$\overline{IRQ}$ and $\overline{NMI}$ require 3K pull-up resistor.

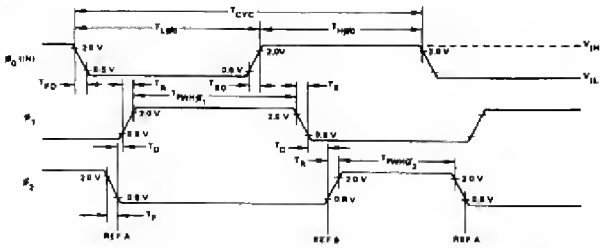
R65XX Timing



Characteristic	Symbol	1 MHz		2 MHz		3 MHz		Unit
		Min	Max	Min	Max	Min	Max	
R/$\overline{W}$ Setup Time	T_{RWS}	–	225	–	140	–	110	ns
R/$\overline{W}$ Hold Time	T_{HRW}	30	–	30	–	15	–	ns
Address Setup Time	T_{ADS}	–	225	–	140	–	110	ns
Address Hold Time	T_{HA}	30	–	30	–	15	–	ns
Read Access Time	T_{ACC}	–	650	–	310	–	170	ns
Read Data Setup Time	T_{DSU}	50	–	40	–	35	–	ns
Read Data Hold Time	T_{HR}	10	–	10	–	10	–	ns
Write Data Setup Time	T_{MDS}	–	175	–	100	–	85	ns
Write Data Hold Time	T_{HW}	30	–	30	–	30	–	ns
SYNC Hold Time	T_{SYH}	30	–	30	–	15	–	ns
RDY Setup Time*	T_{RDY}	100	–	50	–	35	–	ns
S.O. Setup Time	T_{SO}	100	–	50	–	35	–	ns
SYNC Setup Time	T_{SYN}	–	225	–	175	–	100	ns

NOTE
*RDY must never switch states within T_{RDY} to end of $\phi 2$
LOAD = 130 pF + 1 TTL

R650X CPU Clock Timing



Characteristic	Symbol	1 MHz		2 MHz		3 MHz		Unit
		Min	Max	Min	Max	Min	Max	
Cycle Time	T_{CYC}	1.0	10	0.5	10	0.33	10	µs
$\phi_{o(In)}$ Low Time	$T_{L\phi O}$	480	–	240	–	160	–	ns
$\phi_{o(In)}$ High Time	$T_{H\phi O}$	470	–	240	–	160	–	ns
$\phi_{o(In)}$ Rise and Fall Time*	T_{RO}, T_{FO}	–	10	–	10	–	10	ns
ϕ_1 Pulse Width	$T_{PWH\phi_1}$	460	–	235	–	155	–	ns
ϕ_2 Pulse Width	$T_{PWH\phi_2}$	470	–	240	–	160	–	ns
Delay Between ϕ_1 and ϕ_2	T_D	0	–	0	–	0	–	ns
ϕ_1, ϕ_2 Rise and Fall Time*	T_R, T_F	–	25	–	25	–	15	ns

NOTE
*Measured between 0.8 and 2.0 points on waveform load 130 pF + 1 TTL.

Table C.1 6502 Timing Comparisons.

PARAMETER	SYMBOL	SYNERTEK		ROCKWELL		MOS TECHNOLOGY			
		MIN	MAX	MIN	MAX	MIN	TYP	MAX	UNITS
CYCLE TIME	TCYC	.5Ø	4Ø	.5Ø	1Ø	.5Ø	-	---	us
PHSØ LOW TIME	TLØ	24Ø(2)	---	24Ø	---	---	-	---	ns
PHSØ HIGH TIME	THØ	24Ø(2)	---	24Ø	---	---	-	---	ns
PHSØ PULSE WIDTH	PWHØ	---	---	---	---	24Ø(7)	-	26Ø(7)	ns
PHSØ RISE/FALL TIME	TRØ/TFØ	Ø(1)	2Ø	---	1Ø(6)	---	-	1Ø	ns
PHS1 PULSE WIDTH	TPWH1	TLØ-2Ø	TLØ	235	---	TLØ-2Ø(7)	-	TLØ(7)	ns
PHS2 PULSE WIDTH	TPWH2	TLØ-4Ø	TLØ-1Ø	24Ø	---	TLØ-4Ø(7)	-	TLØ-1Ø(7)	ns
DELAY BETWEEN PHS1 AND PHS2	TD	5	---	Ø	---	5(7)	-	---	ns
PHS1, PHS2 RISE/FALL TIMES	TR/TF	---	25(1)(3)	---	25(6)	---	-	25(8)	ns
PHSØ NEG TO PHS1 POS DELAY	TØ1+	1Ø(5)	7Ø(5)	---	---	---	-	---	ns
PHSØ NEG TO PHS2 NEG DELAY	TØ2-	5(5)	65(5)	---	---	---	-	---	ns
PHSØ POS TO PHS1 NEG DELAY	TØ1-	5(5)	65(5)	---	---	---	-	---	ns
PHSØ POS TO PHS2 POS DELAY	TØ2+	15(5)	75(5)	---	---	---	-	---	ns
R/W' SETUP TIME	TRWS	---	14Ø	---	14Ø	---	1ØØ	15Ø	ns
R/W' HOLD TIME	TRWH	3Ø	---	3Ø	---	3Ø	6Ø	---	ns
ADDRESS SETUP TIME	TADS	---	14Ø	---	14Ø	---	1ØØ	15Ø	ns
ADDRESS HOLD TIME	TADH	3Ø	---	3Ø	---	3Ø	6Ø	---	ns
READ ACCESS TIME	TACC	---	31Ø	---	31Ø	---	-	3ØØ	ns
READ DATA SETUP TIME	TDSU	5Ø	---	4Ø	---	5Ø	-	---	ns
READ DATA HOLD TIME	THR	1Ø	---	1Ø	---	1Ø	-	---	ns
WRITE DATA SETUP TIME	TMDS	2Ø	1ØØ	---	1ØØ	---	75	1ØØ	ns
WRITE DATA HOLD TIME	THW	6Ø	15Ø	3Ø	---	3Ø	6Ø	---	ns
SYNC SETUP TIME	TSYS	---	175	---	175	---	-	175	ns
SYNC HOLD TIME	TSYH	3Ø	---	3Ø	---	---	-	---	ns
RDY SETUP TIME	TRS	2ØØ(4)	---	5Ø(4)	---	5Ø(4)	-	---	ns
S.O. SETUP TIME	TSO	---	---	5Ø	---	5Ø	-	---	ns

(1) Measured between 1Ø% and 9Ø% points on waveform.
(2) Measured at 5Ø% points.
(3) Load = 1 TTL load +3Ø pF.
(4) RDY must never switch states within TRS to end of PHS2.
(5) Load = 1ØØ pF.
(6) Measured between .8v and 2.Øv points on waveform, load 13Ø pF + 1 TTL.
(7) Measured at 1.5v.
(8) Measured .8v to 2.Øv, Load 1/2 3ØpF 1/3 1 TTL.

Table C.2 6502 Instruction Execution Periods in Machine Cycles.

	IMP	REL	IMM	ACC	ØPG	ØPG X	ØPG Y	ABS	ABS X	ABS Y	IND	IND X	IND Y
ADC AND CMP EOR LDA ORA SBC			2		3	4		4	4*	4*		6	5*
ASL LSR ROL ROR				2	5	6		6	7				
BCC BCS BEQ BMI BNE BPL BVC BVS		2+p **											
CLC CLD CLI CLV DEX DEY INX INY NOP SEC SED SEI TAX TAY TSX TXA TXS TYA	2												
BIT					3			4					
BRK	7												
CPX CPY			2		3			4					
DEC INC					5	6		6	7				
JMP								3			5		
JSR								6					
LDX			2		3		4	4		4*			
LDY			2		3	4		4	4*				
PHA PHP	3												
PLA PLP	4												
RTI	6												
RTS	6												
STA					3	4		4	5	5		6	6
STX					3		4	4					
STY					3	4		4					

* +1 cycle if indexing crosses page boundary.

** p=0 if branch does not occur.
p=1 if branch within page occurs.
p=2 if branch across page boundary occurs.

■ SIGNAL DESCRIPTION*

Address Bus (A0-A15)
A0-A15 forms a 16-bit address bus for memory and I/O exchanges on the data bus. The output of each address line is TTL compatible, capable of driving one standard TTL load and 130pF.

Clocks (ϕ_0, ϕ_1, and ϕ_2)
ϕ_0 is a TTL level input that is used to generate the internal clocks in the 6502. Two full level output clocks are generated by the 6502. The ϕ_2 clock output is in phase with ϕ_0. The ϕ_1 output pin is 180° out of phase with ϕ_0. (See timing diagram.)

Data Bus (D0-D7)
The data lines (D0-D7) constitute an 8-bit bidirectional data bus used for data exchanges to and from the device and peripherals. The outputs are three-state buffers capable of driving one TTL load and 130 pF.

Interrupt Request ($\overline{IRQ}$)
This TTL compatible input requests that an interrupt sequence begin within the microprocessor. The $\overline{IRQ}$ is sampled during ϕ_2 operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during ϕ_1. The program counter and processor status register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further $\overline{IRQ}$s may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A 3K ohm external resistor should be used for proper wire OR operation.

Memory Lock ($\overline{ML}$)
In a multiprocessor system, the $\overline{ML}$ output indicates the need to defer the rearbitration of the next bus cycle to ensure the integrity of read-modify-write instructions. $\overline{ML}$ goes low during ASL, DEC, INC, LSR, ROL, ROR, TRB, TSB memory referencing instructions. This signal is low for the modify and write cycles.

Non-Maskable Interrupt ($\overline{NMI}$)
A negative-going edge on this input requests that a non-maskable interrupt sequence be generated within the microprocessor. The $\overline{NMI}$ is sampled during ϕ_2; the current instruction is completed and the interrupt sequence begins during ϕ_1. The program counter is loaded with the interrupt vector from locations FFFA (low byte) and FFFB (high byte), thereby transferring program control to the non-maskable interrupt routine.

Note: Since this interrupt is non-maskable, another $\overline{NMI}$ can occur before the first is finished. Care should be taken when using $\overline{NMI}$ to avoid this.

Ready (RDY)
This input allows the user to single-cycle the microprocessor on all cycles including write cycles. A negative transition to the low state, during or coincident with phase one (ϕ_1), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two (ϕ_2) in which the ready signal is low. This feature allows microprocessor interfacing with low-speed memory as well as direct memory access (DMA).

Reset ($\overline{RES}$)
This input is used to reset the microprocessor. Reset must be held low for at least two clock cycles after V_{DD} reaches operating voltage from a power down. A positive transistion on this pin will then cause an initialization sequence to begin. Likewise, after the system has been operating, a low on this line of at least two cycles will cease microprocessing activity, followed by initialization after the positive edge on $\overline{RES}$.

When a positive edge is detected, there is an initialization sequence lasting six clock cycles. Then the interrupt mask flag is set, the decimal mode is cleared, and the program counter is loaded with the restart vector from locations FFFC (low byte) and FFFD (high byte). This is the start location for program control. This input should be high in normal operation.

Read/Write (R/$\overline{W}$)
This signal is normally in the high state indicating that the microprocessor is reading data from memory or I/O bus. In the low state the data bus has valid data from the microprocessor to be stored at the addressed memory location.

Set Overflow ($\overline{SO}$)
A negative transition on this line sets the overflow bit in the status code register. The signal is sampled on the trailing edge of ϕ_1.

Synchronize (SYNC)
This output line is provided to identify those cycles during which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during ϕ_1 of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the ϕ_1 clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

*These signal descriptions, taken directly from the NCR 65C02 specifications, are also accurate for 6502 signals.

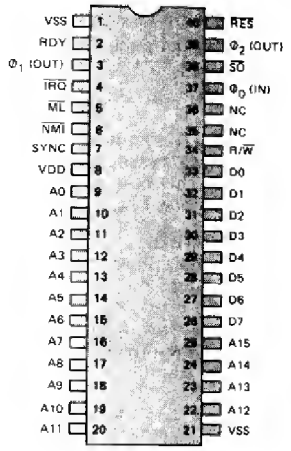
NCR

NCR65C02

▪ GENERAL DESCRIPTION

The NCR CMOS 6502 is an 8-bit microprocessor which is software compatible with the NMOS 6502. The NCR65C02 hardware interfaces with all 6500 peripherals. The enhancements include eight additional instructions, expanded operational codes and two new addressing modes. This microprocessor has all of the advantages of CMOS technology: low power consumption, increased noise immunity and higher reliability. The CMOS 6502 is a low power high performance microprocessor with applications in the consumer, business, automotive and communications market.

▪ PIN CONFIGURATION

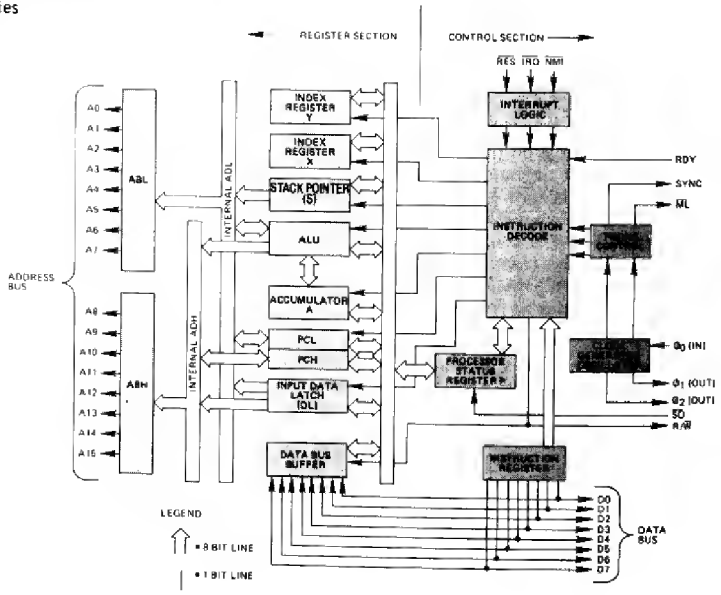


▪ FEATURES

- Enhanced software performance including 27 additional OP codes encompassing ten new instructions and two additional addressing modes.
- 66 microprocessor instructions.
- 15 addressing modes.
- 178 operational codes.
- 1MHz, 2MHz operation.
- Operates at frequencies as low as 200 HZ for even lower power consumption (pseudo-static: stop during $\emptyset_2$ high).
- Compatible with NMOS 6500 series microprocessors.
- 64 K-byte addressable memory.
- Interrupt capability.
- Lower power consumption. 4mA @ 1MHz.
- +5 volt power supply.
- 8-bit bidirectional data bus.
- Bus Compatible with M6800.
- Non-maskable interrupt.
- 40 pin dual-in-line packaging.
- 8-bit parallel processing
- Decimal and binary arithmetic.
- Pipeline architecture.
- Programmable stack pointer.
- Variable length stack.
- Optional internal pullups for (RDY, IRQ, SO, NMI and RES)

* Specifications are subject to change without notice.

▪ NCR65C02 BLOCK DIAGRAM



NCR65C02

■ ABSOLUTE MAXIMUM RATINGS: (V_{DD} = 5.0 V ± 5%, V_{SS} = 0 V, T_A = 0° to + 70°C)

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	V_{DD}	−0.3 to +7.0	V
INPUT VOLTAGE	V_{IN}	−0.3 to +7.0	V
OPERATING TEMP.	T_A	0 to + 70	°C
STORAGE TEMP.	T_{STG}	−55 to + 150	°C

■ PIN FUNCTION

PIN	FUNCTION
A0 - A15	Address Bus
D0 - D7	Data Bus
$\overline{IRQ}$ *	Interrupt Request
RDY *	Ready
$\overline{ML}$	Memory Lock
$\overline{NMI}$ *	Non-Maskable Interrupt
SYNC	Synchronize
$\overline{RES}$ *	Reset
$\overline{SO}$ *	Set Overflow
NC	No Connection
R/$\overline{W}$	Read/Write
VDD	Power Supply (+5V)
VSS	Internal Logic Ground
$\emptyset_0$	Clock Input
$\emptyset_1, \emptyset_2$	Clock Output

*This pin has an optional internal pullup for a No Connect condition.

■ DC CHARACTERISTICS

	SYMBOL	MIN.	TYP.	MAX	UNIT
Input High Voltage $\emptyset_0$ (IN)	V_{IH}	V_{SS} + 2.4	–	V_{DD}	V
Input High Voltage $\overline{RES}$, $\overline{NMI}$, RDY, $\overline{IRQ}$, Data, S.O.		V_{SS} + 2.0	–	–	V
Input Low Voltage $\emptyset_0$ (IN)	V_{IL}	V_{SS} −0.3	–	V_{SS} + 0.4	V
$\overline{RES}$, $\overline{NMI}$, RDY, $\overline{IRQ}$, Data, S.O.		–	–	V_{SS} + 0.8	V
Input Leakage Current (V_{IN} = 0 to 5.25V, V_{DD} = 5.25V)	I_{IN}				
With pullups		−30	–	+30	μA
Without pullups		–	–	+1.0	μA
Three State (Off State) Input Current (V_{IN} = 0.4 to 2.4V, V_{CC} = 5.25V) Data Lines	I_{TSI}	–	–	10	μA
Output High Voltage (I_{OH} = −100 μAdc, V_{DD} = 4.75V SYNC, Data, A0-A15, R/W)	V_{OH}	V_{SS} + 2.4	–	–	V
Out Low Voltage (I_{OL} = 1.6mAdc, V_{DD} = 4.75V SYNC, Data, A0-A15, R/W)	V_{OL}	–	–	V_{SS} + 0.4	V
Supply Current f = 1MHz	I_{DD}	–	–	4	mA
Supply Current f = 2MHz	I_{DD}	–	–	8	mA
Capacitance (V_{IN} = 0, T_A = 25°C, f = 1MHz)	C				pF
Logic	C_{IN}	–	–	5	
Data		–	–	10	
A0-A15, R/W, SYNC	Cout	–	–	10	
$\emptyset_0$ (IN)	$C\emptyset_0$ (IN)	–	–	10	

NCR65C02

■ AC CHARACTERISTICS V_{DD} = 5.0V ± 5%, T_A = 0°C to 70°C, Load = 1 TTL + 130 pF

		1MHz		2MHz		3MHz		
Parameter	Symbol	Min	Max	Min	Max	Min	Max	Unit
Delay Time, $\emptyset_0$ (IN) to $\emptyset_2$ (OUT)	t_{DLY}	–	60	–	60	20	60	nS
Delay Time, $\emptyset_1$ (OUT) to $\emptyset_2$ (OUT)	t_{DLY1}	–20	20	–20	20	–20	20	nS
Cycle Time	t_{CYC}	1.0	5000*	0.50	5000*	0.33	5000*	µS
Clock Pulse Width Low	t_{PL}	460	–	220	–	160	–	nS
Clock Pulse Width High	t_{PH}	460	–	220	–	160	–	nS
Fall Time, Rise Time	t_F, t_R	–	25	–	25	–	25	nS
Address Hold Time	t_{AH}	20	–	20	–	0	–	nS
Address Setup Time	t_{ADS}	–	225	–	140	–	110	nS
Access Time	t_{ACC}	650	–	310	–	170	–	nS
Read Data Hold Time	t_{DHR}	10	–	10	–	10	–	nS
Read Data Setup Time	t_{DSU}	100	–	60	–	60	–	nS
Write Data Delay Time	t_{MDS}	–	30	–	30	–	30	nS
Write Data Hold Time	t_{DHW}	20	–	20	–	15	–	nS
$\overline{SO}$ Setup Time	t_{SO}	100	–	100	–	100	–	nS
Processor Control Setup Time**	t_{PCS}	200	–	150	–	150	–	nS
SYNC Setup Time	t_{SYNC}	–	225	–	140	–	100	nS
$\overline{ML}$ Setup Time	t_{ML}	–	225	–	140	–	100	nS
Input Clock Rise/Fall Time	$t_{F\emptyset_0}$, $t_{R\emptyset_0}$	–	25	–	25	–	25	nS

*NCR65C02 can be held static with $\emptyset_2$ high.

**This parameter must only be met to guarantee that the signal will be recognized at the current clock cycle.

■ MICROPROCESSOR OPERATIONAL ENHANCEMENTS

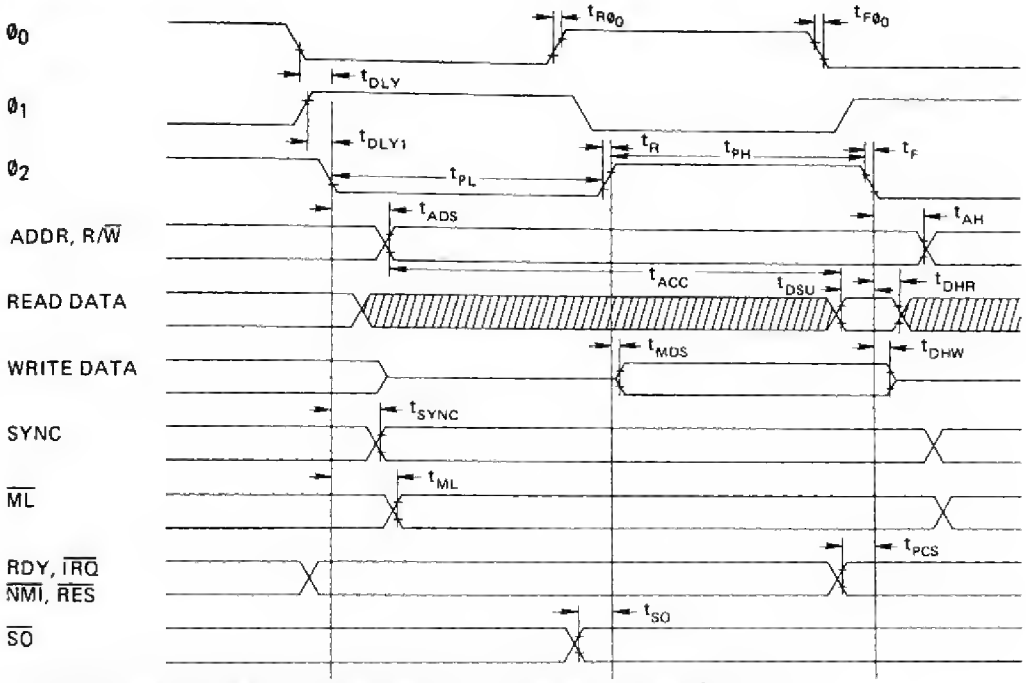
Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). Op Code / Bytes / Cycles X2 / 2 / 2 X3, X7, XB, XF / 1 / 1 44 / 2 / 3 54, D4, F4 / 2 / 4 5C / 3 / 8 DC, FC / 3 / 4
Jump indirect, operand = XXFF.	Page address does not increment.	Page address increments and adds one additional cycle.
Read/modify/write instructions at effective address.	One read and two write cycles.	Two read and one write cycle.
Decimal flag.	Indeterminate after reset.	Initialized to binary mode (D=0) after reset and interrupts.
Flags after decimal operation.	Invalid N, V and Z flags.	Valid flag adds one additional cycle.
Interrupt after fetch of BRK instruction.	Interrupt vector is loaded, BRK vector is ignored.	BRK is executed, then interrupt is executed.

■ MICROPROCESSOR HARDWARE ENHANCEMENTS

Function	NMOS 6502	NCR65C02
Assertion of Ready RDY during write operations.	Ignored.	Stops processor during $\emptyset_2$.
Unused input-only pins ($\overline{IRQ}$, $\overline{NMI}$, RDY, $\overline{RES}$, $\overline{SO}$).	Must be connected to low impedance signal to avoid noise problems.	Connected internally by a high-resistance to V_{DD} (approximately 250 K ohm.)

NCR65C02

▪ TIMING DIAGRAM



Note: All timing is referenced from a high voltage of 2.0 volts and a low voltage of 0.8 volts.

▪ NEW INSTRUCTION MNEMONICS

HEX	MNEMONIC	DESCRIPTION
80	BRA	Branch relative always [Relative]
3A	DEA	Decrement accumulator [Accum]
1A	INA	Increment accumulator [Accum]
DA	PHX	Push X on stack [Implied]
5A	PHY	Push Y on stack [Implied]
FA	PLX	Pull X from stack [Implied]
7A	PLY	Pull Y from stack [Implied]
9C	STZ	Store zero [Absolute]
9E	STZ	Store zero [ABS, X]
64	STZ	Store zero [Zero page]
74	STZ	Store zero [ZPG,X]
1C	TRB	Test and reset memory bits with accumulator [Absolute]
14	TRB	Test and reset memory bits with accumulator [Zero page]
0C	TSB	Test and set memory bits with accumulator [Absolute]
04	TSB	Test and set memory bits with accumulator [Zero page]

▪ ADDITIONAL INSTRUCTION ADDRESSING MODES

HEX	MNEMONIC	DESCRIPTION
72	ADC	Add memory to accumulator with carry [(ZPG)]
32	AND	"AND" memory with accumulator [(ZPG)]
3C	BIT	Test memory bits with accumulator [ABS, X]
34	BIT	Test memory bits with accumulator [ZPG, X]
D2	CMP	Compare memory and accumulator [(ZPG)]
52	EOR	"Exclusive Or" memory with accumulator [(ZPG)]
7C	JMP	Jump (New addressing mode) [ABS(IND,X)]
B2	LDA	Load accumulator with memory [(ZPG)]
12	ORA	"OR" memory with accumulator [(ZPG)]
F2	SBC	Subtract memory from accumulator with borrow [(ZPG)]
92	STA	Store accumulator in memory [(ZPG)]

NCR65C02
▪ ADDRESSING MODES

Fifteen addressing modes are available to the user of the NCR65C02 microprocessor. The addressing modes are described in the following paragraphs:

Implied Addressing [Implied]
In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

Accumulator Addressing [Accum]
This form of addressing is represented with a one byte instruction and implies an operation on the accumulator.

Immediate Addressing [Immediate]
With immediate addressing, the operand is contained in the second byte of the instruction; no further memory addressing is required.

Absolute Addressing [Absolute]
For absolute addressing, the second byte of the instruction specifies the eight low-order bits of the effective address, while the third byte specifies the eight high-order bits. Therefore, this addressing mode allows access to the total 64K bytes of addressable memory.

Zero Page Addressing [Zero Page]
Zero page addressing allows shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. The careful use of zero page addressing can result in significant increase in code efficiency.

Absolute Indexed Addressing [ABS, X or ABS, Y]
Absolute indexed addressing is used in conjunction with X or Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields, resulting in reduced coding and execution time.

Zero Page Indexed Addressing [ZPG, X or ZPG, Y]
Zero page absolute addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally, due to the "Zero Page" addressing nature of this mode, no carry is added to the high-order eight bits of memory, and crossing of page boundaries does not occur.

Relative Addressing [Relative]
Relative addressing is used only with branch instructions; it establishes a destination for the conditional branch. The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the program counter when the counter is set at the next instruction. The range of the offset is −128 to +127 bytes from the next instruction.

Zero Page Indexed Indirect Addressing [(IND, X)]
With zero page indexed indirect addressing (usually referred to as indirect X) the second byte of the instruction is added to the contents of the X index register; the carry is discarded. The result of this addition points to a memory location on page zero whose contents is the low-order eight bits of the effective address. The next memory location in page zero contains the high-order eight bits of the effective address. Both memory locations specifying the high- and low-order bytes of the effective address must be in page zero.

***Absolute Indexed Indirect Addressing [ABS(IND, X)] (Jump Instruction Only)**
With absolute indexed indirect addressing the contents of the second and third instruction bytes are added to the X register. The result of this addition, points to a memory location containing the lower-order eight bits of the effective address. The next memory location contains the higher-order eight bits of the effective address.

Indirect Indexed Addressing [(IND), Y]
This form of addressing is usually referred to as Indirect, Y. The second byte of the instruction points to a memory location in page zero. The contents of this memory location are added to the contents of the Y index register, the result being the low-order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high-order eight bits of the effective address.

***Zero Page Indirect Addressing [(ZPG)]**
In the zero page indirect addressing mode, the second byte of the instruction points to a memory location on page zero containing the low-order byte of the effective address. The next location on page zero contains the high-order byte of the effective address.

Absolute Indirect Addressing [(ABS)] (Jump Instruction Only)
The second byte of the instruction contains the low-order eight bits of a memory location. The high-order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low-order byte of the effective address. The next memory location contains the high-order byte of the effective address which is loaded into the 16 bit program counter.

NOTE: * = New Address Modes

NCR65C02
▪ INSTRUCTION SET — ALPHABETICAL SEQUENCE

ADC Add Memory to Accumulator with Carry
AND "AND" Memory with Accumulator
ASL Shift One Bit Left
BCC Branch on Carry Clear
BCS Branch on Carry Set
BEQ Branch on Result Zero
BIT Test Memory Bits with Accumulator
BMI Branch on Result Minus
BNE Branch on Result not Zero
BPL Branch on Result Plus
*BRA Branch Always
BRK Force Break
BVC Branch on Overflow Clear
BVS Branch on Overflow Set
CLC Clear Carry Flag
CLD Clear Decimal Mode
CLI Clear Interrupt Disable Bit
CLV Clear Overflow Flag
CMP Compare Memory and Accumulator
CPX Compare Memory and Index X
CPY Compare Memory and Index Y
*DEA Decrement Accumulator
DEC Decrement by One
DEX Decrement Index X by One
DEY Decrement Index Y by One
EOR "Exclusive-or" Memory with Accumulator
*INA Increment Accumulator
INC Increment by One
INX Increment Index X by One
INY Increment Index Y by One
JMP Jump to New Location
JSR Jump to New Location Saving Return Address
LDA Load Accumulator with Memory
LDX Load Index X with Memory
LDY Load Index Y with Memory
LSR Shift One Bit Right
NOP No Operation
ORA "OR" Memory with Accumulator
PHA Push Accumulator on Stack
PHP Push Processor Status on Stack
*PHX Push Index X on Stack
*PHY Push Index Y on Stack
PLA Pull Accumulator from Stack
PLP Pull Processor Status from Stack
*PLX Pull Index X from Stack
*PLY Pull Index Y from Stack
ROL Rotate One Bit Left
ROR Rotate One Bit Right
RTI Return from Interrupt
RTS Return from Subroutine
SBC Subtract Memory from Accumulator with Borrow
SEC Set Carry Flag
SED Set Decimal Mode
SEI Set Interrupt Disable Bit
STA Store Accumulator in Memory
STX Store Index X in Memory
STY Store Index Y in Memory
*STZ Store Zero in Memory
TAX Transfer Accumulator to Index X
TAY Transfer Accumulator to Index Y
*TRB Test and Reset Memory Bits with Accumulator
*TSB Test and Set Memory Bits with Accumulator
TSX Transfer Stack Pointer to Index X
TXA Transfer Index X to Accumulator
TXS Transfer Index X to Stack Pointer
TYA Transfer Index Y to Accumulator

Note: * = New Instruction

▪ MICROPROCESSOR OP CODE TABLE

S D	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK	ORA ind, X			TSB* zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB* abs	ORA abs	ASL abs		0
1	BPL rel	ORA ind, Y	ORA*† (zpg)		TRB* zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INA* A		TRB* abs	ORA abs, X	ASL abs, X		1
2	JSR abs	AND ind, X			BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs		2
3	BMI rel	AND ind, Y	AND*† (zpg)		BIT* zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEA* A		BIT*† abs, X	AND abs, X	ROL abs, X		3
4	RTI	EOR ind, X				EOR zpg	LSR zpg		PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs		4
5	BVC rel	EOR ind, Y	EOR*† (zpg)			EOR zpg, X	LSR zpg, X		CLI	EOR abs, Y	PHY*			EOR abs, X	LSR abs, X		5
6	RTS	ADC ind, X			STZ* zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP (abs)	ADC abs	ROR abs		6
7	BVS rel	ADC ind, Y	ADC*† (zpg)		STZ* zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY*		JMP*† abs (ind, X)	ADC abs, X	ROR abs, X		7
8	BRA* rel	STA ind, X			STY zpg	STA zpg	STX zpg		DEY	BIT* imm	TXA		STY abs	STA abs	STX abs		8
9	BCC rel	STA ind, Y	STA*† (zpg)		STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS		STZ* abs	STA abs, X	STZ* abs, X		9
A	LDY imm	LDA ind, X	LDX imm		LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX		LDY abs	LDA abs	LDX abs		A
B	BCS rel	LDA ind, Y	LDA*† (zpg)		LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX		LDY abs, X	LDA abs, X	LDX abs, Y		B
C	CPY imm	CMP ind, X			CPY zpg	CMP zpg	DEC zpg		INY	CMP imm	DEX		CPY abs	CMP abs	DEC abs		C
D	BNE rel	CMP ind, Y	CMP*† (zpg)			CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX*			CMP abs, X	DEC abs, X		D
E	CPX imm	SBC ind, X			CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP		CPX abs	SBC abs	INC abs		E
F	BEQ rel	SBC ind, Y	SBC*† (zpg)			SBC zpg, X	INC zpg, X		SED	SBC abs, Y	PLX*			SBC abs, X	INC abs, X		F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Note: * = New OP Codes
Note: † = New Address Modes

NCR65C02

▪ OPERATIONAL CODES, EXECUTION TIME, AND MEMORY REQUIREMENTS

MNE	OPERATION		IMMEDIATE OP	n	#	ABSOLUTE OP	n	#	ZERO PAGE OP	n	#	ACCUM OP	n	#	IMPLIED OP	n	#	(IND, X) OP	n	#	(IND), Y OP	n	#	ZPG, X OP	n	#	ZPG, Y OP	n	#	ABS, X OP	n	#	ABS, Y OP	n	#	RELATIVE OP	n	#	(ABS) OP	n	#	ABS (IND, X) OP	n	#	(ZPG) OP	n	#	PROCESSOR STATUS CODES 7 6 5 4 3 2 1 0	MNE
ADC	A + M + C → A	(1,3)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2				7D	4	3	79	4	3											72	5	2	N V . . . . Z C	ADC
AND	A ∧ M → A	(1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2				3D	4	3	39	4	3											32	5	2	N . . . . . Z .	AND
ASL	C ← 7 ... 0 ← 0	(1)				0E	6	3	06	5	2	0A	2	1										16	6	2				1E	6	3														N . . . . . Z C	ASL
BCC	Branch if C=0	(2)																																			90	2	2									. . . . .	BCC
BCS	Branch if C=1	(2)																																			B0	2	2									. . . . .	BCS
BEQ	Branch if Z=1	(2)																																			F0	2	2									. . . . .	BEQ
BIT	A ∧ M	(4,5)	89	2	2	2C	4	3	24	3	2													34	4	2				3C	4	3																M_7 M_6 . . . . Z .	BIT
BMI	Branch if N=1	(2)																																			30	2	2									. . . . .	BMI
BNE	Branch if Z=0	(2)																																			D0	2	2									. . . . .	BNE
BPL	Branch if N=0	(2)																																			10	2	2									. . . . .	BPL
BRA	Branch Always	(2)																																			80	2	2									. . . . .	BRA
BRK	Break														00	7	1																															. . . 1 . .	BRK
BVC	Branch if V=0	(2)																																			50	2	2									. . . . .	BVC
BVS	Branch if V=1	(2)																																			70	2	2									. . . . .	BVS
CLC	0 → C														18	2	1																															 0	CLC
CLD	0 → D														D8	2	1																															. . . 0 . .	CLD
CLI	0 → I														58	2	1																															. . . . 0 . .	CLI
CLV	0 → V														B8	2	1																															0	CLV
CMP	A − M	(1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2				DD	4	3	D9	4	3											D2	5	2	N . . . . . Z C	CMP
CPX	X − M		E0	2	2	EC	4	3	E4	3	2																																					N . . . . . Z C	CPX
CPY	Y − M		C0	2	2	CC	4	3	C4	3	2																																					N . . . . . Z C	CPY
DEA	A − 1 → A											3A	2	1																																	N . . . . . Z .	DEA
DEC	M − 1 → M	(1)				CE	6	3	C6	5	2													D6	6	2				DE	6	3																N . . . . . Z .	DEC
DEX	X − 1 → X														CA	2	1																															N . . . . . Z .	DEX
DEY	Y − 1 → Y														88	2	1																															N . . . . . Z .	DEY
EOR	A ⊻ M → A		49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2				5D	4	3	59	4	3											52	5	2	N . . . . . Z .	EOR
INA	A + 1 → A											1A	2	1																																	N . . . . . Z .	INA
INC	M + 1 → M	(1)				EE	6	3	E6	5	2													F6	6	2				FE	6	3																N . . . . . Z .	INC
INX	X + 1 → X														E8	2	1																															N . . . . . Z .	INX
INY	Y + 1 → Y														C8	2	1																															N . . . . . Z .	INY
JMP	Jump to new loc					4C	3	3																																	6C	6	3	7C	6	3				. . . . .	JMP
JSR	Jump Subroutine					20	6	3																																								. . . . .	JSR
LDA	M → A	(1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2				BD	4	3	B9	4	3											B2	5	2	N . . . . . Z .	LDA
LDX	M → X	(1)	A2	2	2	AE	4	3	A6	3	2																B6	4	2				BE	4	3													N . . . . . Z .	LDX
LDY	M → Y	(1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2				BC	4	3																N . . . . . Z .	LDY
LSR	0 → 7 ... 0 → C	(1)				4E	6	3	46	5	2	4A	2	1										56	6	2				5E	6	3																0 . . . . . Z C	LSR
NOP	PC + 1 → PC														EA	2	1																															. . . . .	NOP
ORA	A ∨ M → A	(1)	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2				1D	4	3	19	4	3											12	5	2	N . . . . . Z .	ORA
PHA	A → M_S, S − 1 → S														48	3	1																															. . . . .	PHA
PHP	P → M_S, S − 1 → S														08	3	1																															. . . . .	PHP
PHX	X → M_S, S − 1 → S														DA	3	1																															. . . . .	PHX
PHY	Y → M_S, S − 1 → S														5A	3	1																															. . . . .	PHY
PLA	S + 1 → S, M_S → A														68	4	1																															N . . . . . Z .	PLA
PLP	S + 1 → S, M_S → P														28	4	1																															N V . 1 D I Z C	PLP
PLX	S + 1 → S, M_S → X														FA	4	1																															N . . . . . Z .	PLX
PLY	S + 1 → S, M_S → Y														7A	4	1																															N . . . . . Z .	PLY
ROL	C ← 7 ... 0 ← C	(1)				2E	6	3	26	5	2	2A	2	1										36	6	2				3E	6	3																N . . . . . Z C	ROL
ROR	C → 7 ... 0 → C	(1)				6E	6	3	66	5	2	6A	2	1										76	6	2				7E	6	3																N . . . . . Z C	ROR
RTI	Return from Inter														40	6	1																															N V . 1 D I Z C	RTI
RTS	Return from Subr														60	6	1																															. . . . .	RTS
SBC	A − M − $\overline{C}$ → A	(1,3)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2				FD	4	3	F9	4	3											F2	5	2	N V . . . . Z C	SBC
SEC	1 → C														38	2	1																															 1	SEC
SED	1 → D														F8	2	1																															. . . 1 . .	SED
SEI	1 → I														78	2	1																															. . . . 1 . .	SEI
STA	A → M					8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2				9D	5	3	99	5	3											92	5	2	. . . . .	STA
STX	X → M					8E	4	3	86	3	2																96	4	2																					. . . . .	STX
STY	Y → M					8C	4	3	84	3	2													94	4	2																								. . . . .	STY
STZ	00 → M					9C	4	3	64	3	2													74	4	2				9E	5	3																	. . . . .	STZ
TAX	A → X														AA	2	1																															N . . . . . Z .	TAX
TAY	A → Y														A8	2	1																															N . . . . . Z .	TAY
TRB	A ∧ M → M	(4)				1C	6	3	14	5	2																																					. . . . . Z .	TRB
TSB	A ∨ M → M	(4)				0C	6	3	04	5	2																																					. . . . . Z .	TSB
TSX	S → X														BA	2	1																															N . . . . . Z .	TSX
TXA	X → A														8A	2	1																															N . . . . . Z .	TXA
TXS	X → S														9A	2	1																															. . . . .	TXS
TYA	Y → A														98	2	1																															N . . . . . Z .	TYA

Notes:

1. Add 1 to "n" if page boundary is crossed.
2. Add 1 to "n" if branch occurs to same page.
   Add 2 to "n" if branch occurs to different page.
3. Add 1 to "n" if decimal mode.
4. V bit equals memory bit 6 prior to execution.
   N bit equals memory bit 7 prior to execution.
5. *The immediate addressing mode of the BIT instruction leaves bits 6 & 7 (V & N) in the Processor Status Code Register unchanged.

X Index X
Y Index Y
A Accumulator
M Memory per effective address
Ms Memory per stack pointer

\+ Add
− Subtract
∧ And
∨ Or
⊻ Exclusive or

n No. Cycles
No. Bytes
M_6 Memory bit 6
M_7 Memory bit 7

Table C.3 65C02 Instruction Execution Periods in Machine Cycles.

	IMP	REL	IMM	ACC	ØPG	ØPG X	ØPG Y	ABS	ABS X	ABS Y	ABS IND	ØPG IND X	ØPG IND Y	ØPG IND	ABS IND X	ØPG REL
ADC AND CMP EOR LDA ORA SBC ***			2		3	4		4	4*	4*		6	5*	5		
ASL LSR ROL ROR **DEC INC**				2	5	6		6	**6***							
BBRn BBSn**																5+p **
BCC BCS BEQ BMI BNE BPL BVC BVS **BRA**		2+p **														
CLC CLD CLI CLV DEX DEY INX INY NOP SEC SED SEI TAX TAY TSX TXA TXS TYA	2															
BIT			2		3	4		4	4*							
BRK	7															
CPX CPY			2		3			4								
JMP								3			6				6	
JSR								6								
LDX			2		3		4	4		4*						
LDY			2		3	4		4	4*							
PHA PHP **PHX PHY**	3															
PLA PLP **PLX PLY**	4															
RMBn SMBn**					5											
RTI	6															
RTS	6															
STA					3	4		4	5	5		6	6	5		
STX					3		4	4								
STY					3	4		4								
STZ					3	4		4	5							
TRB TSB					5			6								

NOTES:
* +1 cycle if indexing crosses page boundary.
** p=0 if branch does not occur.
p=1 if branch within page occurs.
p=2 if branch across page boundary occurs.
*** Add 1 cycle to ADC and SBC if decimal mode.
**** BBRn, BBSn, RMBn, and SMBn instructions are available in Rockwell 65C02 but not NCR 65C02.

Boldfaced type indicates difference between 65C02 and 6502.

Unused op codes:

$02, $22, $42, $62, $82, $C2, $E2	=> 2 bytes, 2 cycles
$X3, $XB	=> 1 byte, 1 cycle
$X7, $XF	=> 1 byte, 1 cycle (NCR only)
$44	=> 2 bytes, 3 cycles
$54, $D4, $F4	=> 2 bytes, 4 cycles
$5C	=> 3 bytes, 8 cycles
$DC, $FC	=> 3 bytes, 4 cycles

appendix D

BASIC Program Listings

The following pages contain the BASIC program listings which produce Figures 5.8 and 5.9.

```
0  REM
1  REM
2  REM
3  REM  HIRES MEMORY MAP - DISPLAYED SCAN ONLY.
4  REM
5  REM
6  REM
10  DIM HX$(15)
20  DATA  "0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"
30  FOR A = 0 TO 15: READ HX$(A): NEXT A
40  TEXT : HOME
50  PRINT "                                  TOP SCREEN/         MIDDLE SCREEN/      BOTTOM SCREEN/"
60  PRINT "                                   FIRST 40            SECOND 40           THIRD 40        UNUSED 8"
70  PRINT "   PAGE 1      PAGE 2   LIN# PAGE 1 RANGE   LIN# PAGE 1 RANGE   LIN# PAGE 1 RANGE   PAGE 1 RANGE"
90  REM
91  REM
100  FOR A = 0 TO 7: FOR B = 0 TO 7:BASE = A * 128 + B * 1024 + 8192
110 LN$ = "$":DEC = BASE: GOSUB 5000: REM GET PAGE 1 HEX
190  REM
191  REM
200 DEC$ =  STR$ (BASE): IF  LEN (DEC$) < 5 THEN LN$ = LN$ + " "
210 LN$ = LN$ + " " + DEC$ + " $"
220 DEC = BASE + 8192: GOSUB 5000: REM GET PAGE 2 HEX
230 LN$ = LN$ + " " +  STR$ (DEC) + " "
290  REM
291  REM
300 SCAN = 8 * A + B: GOSUB 6000: REM  GET SCAN NUMBER
310 DEC = BASE: GOSUB 5000
320 LN$ = LN$ + "-$"
330 DEC = BASE + 39: GOSUB 5000
340 LN$ = LN$ + "    "
390  REM
391  REM
400 SCAN = SCAN + 64: GOSUB 6000
410 DEC = BASE + 40: GOSUB 5000
420 LN$ = LN$ + "-$"
430 DEC = BASE + 79: GOSUB 5000
440 LN$ = LN$ + "    "
490  REM
491  REM
500 SCAN = SCAN + 64: GOSUB 6000
510 DEC = BASE + 80: GOSUB 5000
520 LN$ = LN$ + "-$"
530 DEC = BASE + 119: GOSUB 5000
590  REM
591  REM
600 LN$ = LN$ + "     $"
610 DEC = BASE + 120: GOSUB 5000
620 LN$ = LN$ + "-$"
630 DEC = BASE + 127: GOSUB 5000
640  PRINT LN$: NEXT B: NEXT A
650  PRINT : PRINT
660  PRINT "FIGURE 5.8 - HIRES DISPLAYED MEMORY MAP.  PAGE 1 AND PAGE 2 ARE EACH MADE UP OF 64 128-BYTE MEMORY SEGMENTS."
670  GET B$: END
4090  REM
4091  REM
4092  REM
4093  REM  SUBROUTINE 5000 CONVERTS THE DECIMAL ADDRESS IN DEC TO
4094  REM  HEXADECIMAL AND CONCATINATES THE HEX NUMBER TO LN$.
4095  REM
4096  REM
4097  REM
5000 H4 = DEC / 4096:H4% = H4
5010 H3 = (H4 - H4%) * 16:H3% = H3
5020 H2 = (H3 - H3%) * 16:H2% = H2
5030 H1% = (H2 - H2%) * 16 + .5
5040 LN$ = LN$ + HX$(H4%) + HX$(H3%) + HX$(H2%) + HX$(H1%)
5050  RETURN
5190  REM
5191  REM
5192  REM
5193  REM   SUBROUTINE 6000 ADDS LEADING ZEROES TO THE SCAN #
5194  REM
5195  REM
5196  REM
6000  IF SCAN < 100 THEN LN$ = LN$ + "0"
6010  IF SCAN < 10 THEN LN$ = LN$ + "0"
6020 LN$ = LN$ +  STR$ (SCAN) + "  $"
6030  RETURN
```

Figure D.1 BASIC Listing: Program that Produces Figure 5.8.

```
0  REM
3  REM  FOUR PAGE HIRES MEMORY MAP - INCLUDES VBL & HBL.
6  REM
10  DIM HX$(15)
20  DATA  "0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"
30  FOR A = 0 TO 15: READ HX$(A): NEXT A
40 SCAN =  - 1
100  REM
102  PRINT  TAB( 56)"SCREEN TOP": PRINT : PRINT
110  GOSUB 4000: REM  PRINT HEADINGS
120  FOR A = 0 TO 7: FOR B = 0 TO 7:BASE = A * 128 + B * 1024 + 8192
130  GOSUB 1000: REM  DO LINE
140  NEXT B: NEXT A
150  GOSUB 5000: GET A$
200  REM  ********************
202  PRINT : PRINT : PRINT  TAB( 55)"SCREEN MIDDLE": PRINT : PRINT
210  GOSUB 4000
220  FOR A = 0 TO 7: FOR B = 0 TO 7:BASE = A * 128 + B * 1024 + 8232
230  GOSUB 1000
240  NEXT B: NEXT A
250  GOSUB 5000: GET A$
300  REM  ********************
301  PRINT : PRINT : PRINT  TAB( 55)"SCREEN BOTTOM": PRINT : PRINT
310  GOSUB 4000
320  FOR A = 0 TO 7: FOR B = 0 TO 7:BASE = A * 128 + B * 1024 + 8272
330  GOSUB 1000
340  NEXT B: NEXT A
350  GOSUB 5000: GET A$
400  REM  ********************
401  PRINT : PRINT : PRINT  TAB( 46)"VERTICAL BLANKING PERIOD (VBL)": PRINT : PRINT
410  GOSUB 4000
420  FOR A = 0 TO 7: FOR B = 0 TO 7:BASE = A * 128 + B * 1024 + 8312
430  GOSUB 1000
440  NEXT B: NEXT A
450  FOR B = 2 TO 7:BASE = 9208 + B * 1024: REM 7*128+8312+B*1024
460  GOSUB 1000
470  NEXT B
480  GOSUB 5000: GET A$
490  END
1000  REM
1002  REM  PRINT A LINE
1004  REM
1010 LN$ = "":SCAN = SCAN + 1: IF SCAN < 10 THEN LN$ = LN$ + " "
1020  IF SCAN < 100 THEN LN$ = LN$ + " "
1025 LN$ = LN$ +  STR$ (SCAN) + "  "
1030 HBL = BASE - 24: IF  INT (HBL / 128) <  >  INT (BASE / 128) THEN HBL = HBL + 128
1040 DEC = HBL: GOSUB 3000: REM  DO PAGE 1
1050 DEC = HBL + 8192: GOSUB 3000: REM  DO PAGE 2
1060 SYMBOL$$ = "++++++++++++ ": IF SCAN > 223 AND SCAN < 228 THEN SYMBOL$$ = "############ "
1065 LN$ = LN$ + "+++++++++####" + SYMBOL$$
1070 DEC = BASE: GOSUB 3000: REM  DO PAGE 1
1080 DEC = BASE + 8192: GOSUB 3000: REM DO PAGE 2
1090 SYMBOL$$ = "++++++++++++++++++++++++++++++++++++++++"
1100  IF SCAN > 223 AND SCAN < 228 THEN SYMBOL$$ = "########################################"
1110 LN$ = LN$ + SYMBOL$$: PRINT LN$
1120  RETURN
3000  REM
3002  REM  GET HEX & DECIMAL STRINGS
3004  REM
3005 H4 = DEC / 4096:H4% = H4
3010 H3 = (H4 - H4%) * 16:H3% = H3
3020 H2 = (H3 - H3%) * 16:H2% = H2
3030 H1% = (H2 - H2%) * 16 + .5
3040 LN$ = LN$ + "$" + HX$(H4%) + HX$(H3%) + HX$(H2%) + HX$(H1%) + " "
3050 DEC$ =  STR$ (DEC): IF  LEN (DEC$) < 5 THEN LN$ = LN$ + " "
3060 LN$ = LN$ + DEC$ + " "
3070  RETURN
4000  REM
4002  REM  PRINT HEADINGS
4004  REM
4010  PRINT "                 HORIZONTAL BLANKING (HBL)                                HORIZONTAL DISPLAY ENABLE"
4015  PRINT
4020  PRINT "LINE                                        11111111                                  1111111111111111122222222"
4030  PRINT "NUM     PAGE 1      PAGE 2   00123456789ABCDEF01234567    PAGE 1      PAGE 2   0123456789ABCDEF0123456789ABCDEF01234567"
4040  RETURN
5000  REM
5002  REM  PRINT FIGURE NUMBER
5004  REM
5005  PRINT : PRINT
5010  PRINT "FIGURE 5.9 - THIS HIRES SCANNING MAP SHOWS THE ADDRESSES SCANNED DURING DISPLAY PERIODS AND BLANKING PERIODS."
5020  RETURN
```

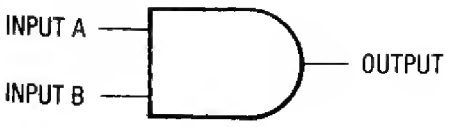
Figure D.2 BASIC Listing: Program that Produces Figure 5.9.

appendix E

A Logic Circuits Primer

Bits of information in a computer are generally represented by voltages. In positive level logic like that used in the Apple, a high voltage (about 3 volts) is considered to be true, and a low voltage (about 0 volts) is considered to be false. The electronic circuits in the Apple are designed primarily to treat signal voltages as true or false indications and to process them logically. In studying the Apple, it is advisable to concentrate on the logical function of the components rather than their electronic function.

The most basic functional building blocks are simple logic gates. For example, a 2-input AND gate will bring its output high if and only if both inputs are high. In other words, both input A AND input B must be true if the output is to be true. The two input AND gate is represented in logic diagrams as follows:



This AND function is identical to the 6502 AND instruction, except that the 6502 instruction is performed on eight bits simultaneously and is logically equivalent to eight 2-input AND gates.

A way of demonstrating a logic function is a truth table. The truth table shows the state of an output for every possible combination of inputs. The true state can be represented by T or 1 or H (for high assuming positive logic), and the false state can be represented by F or 0 or L. We will use H and L because this usually eliminates possible ambiguities. The truth table for the positive logic AND gate is:

INPUT A	INPUT B	OUTPUT
L	L	L
L	H	L
H	L	L
H	H	H

This clearly demonstrates that both inputs of the positive logic AND gate must be high before the output will go high. Table E.1 shows the truth table, schematic representation, and equivalent 6502 instruction, where applicable, of some simple logic gates used in this book.

The little circles on the gates represent the concepts of inversion and active-when-low signals. The two concepts are closely related and sometimes impossible to separate. Inversion is the process of turning a signal into the logically opposite signal. When a signal is low, its inversion is high, and vice versa. When examining a logic gate, the absence of circles can be read as active-when-high and the circles can be read as active-when-low. For example, the NAND gate has a circle on its output, meaning both inputs must go high to make the output go low. This can be stated in a second way. If either input goes low, the output will go high. This results in a second way of representing the NAND gate, as an OR gate with little circles on the inputs. The circles take a little getting used to and are used in some pretty unusual ways in some drawings. Just associate the circle with the word "low" and you should get the message.

The tri-state amplifiers of Table E.1 represent a different sort of logic device. In addition to the normal binary states of high and low voltage, the tri-state device has a third state, high isolation or high impedance. The line coming in from the side is the output enable line and it controls the isolation. When the output enable is not active, the device is isolated from the output line, so another device can control the output line. In electronic terms, the device presents a high impedance to the output line. In the truth tables of Table E.1, the high impedance state is represented by a "Z". A more detailed discussion of tri-state logic is contained in the chapter on bus structure.

A building block of equal importance to the logic gates is the clocked flip-flop. This is a 1-bit storage device which will respond to its logic inputs when it senses an active transition on its clockpulse input. Figure E.1 shows a diagram of a D type flip-flop and its truth table. The flip-flop shown is clocked by a low-to-high transition of its clockpulse input, and is like the 74LS74 flip-flop pictured in Figure 6.13. The Q output will follow the D input every time the clockpulse rises, and the Q' output will be the inversion of the Q output. The CLEAR and PRESET inputs cause the flip-flop to change states without requiring a clock, and actually override the clocked D input. Bringing PRESET low forces Q high and Q' low. Bringing CLEAR low forces Q low and Q' high.

The clockpulse adds synchronization to logic. If the same clockpulse triggers a hundred different actions, then the actions all occur simultaneously. This clockpulse synchronization is common to all digital computers. Certain devices react to the clock. Other devices react to those clocked devices, and so on. After a given period of time, all reactions are complete and the logic signals are all stable, waiting for the next clock. The computer thus operates one cycle at a time.

As an example of clocked operation, Figure E.2 shows a logic function similar to the 6502 AND instruction. In the AND instruction, the value in the accumulator is ANDed with a different value to get the new accumulator value. In Figure E.2, the flip-flop represents one bit of the accumulator. When the flip-flop clock rises, the flip-flop goes to a state determined by its old value ANDed with a second value.

Most logical circuitry is made up of some combination of simple logic gates and flip-flops or their equivalents inside an integrated circuit. In modern computers, many complex functions are available packaged in integrated circuits. Typical of such complex functions are comparison, counting, coding, decoding, and shifting. Even more complex are the functions of chips like the 6502, RAM, ROM, the MMU, and the IOU in the Apple IIe. A good way to familiarize yourself with the variety of logic functions available is to peruse the data books published by manufacturers. Of particular help in the Apple is a TTL data book. TTL (Transistor Transistor Logic) is the name of the logic family to which most of the Apple's general purpose chips belong. National Semiconductor is a company which is very good at making their data books available to the public at reasonable prices. Their TTL data is contained in the *Logic Databook*, priced at $9.00 as of March, 1982. Books can be obtained by writing:

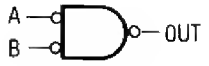
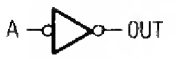
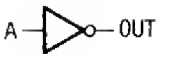
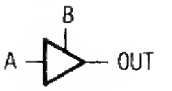
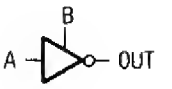
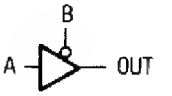
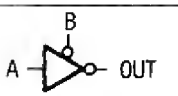
National Semiconductor Corporation
ATTN: Literature Distribution MS/14208
2900 Semiconductor Drive
Santa Clara, CA 95051

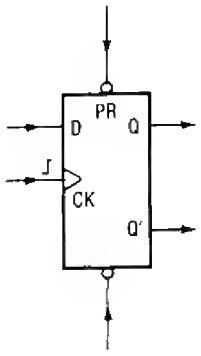
Understanding the Apple IIe uses logic equations to describe the logical makeup of certain signals. A typical logic equation is

$$GR = (TEXT + MIX \bullet V4 \bullet V2)',$$

which is equivalent to GR = NOT (TEXT OR (MIX AND V4 AND V2)). The dot represents the AND function, the plus sign represents the OR function,

Table E.1 Basic Logic Gates.

NAME	REPRESENTATION	SECONDARY REPRESENTATION	TRUTH TABLE B A OUT	6502 EQUIVALENT
AND			L L L L H L H L L H H H	AND XXXX
OR			L L L L H H H L H H H H	ORA XXXX
NAND			L L H L H H H L H H H L	AND XXXX EOR #$FF
NOR			L L H L H L H L L H H L	ORA XXXX EOR #$FF
EXCLUSIVE OR			L L L L H H H L H H H L	EOR XXXX
AMPLIFIER			L L H H	NOP
INVERTER			L H H L	EOR #$FF
TRI-STATE AMPLIFIER HIGH ENABLE			L L Z L H Z H L L H H H	———
TRI-STATE INVERTER HIGH ENABLE			L L Z L H Z H L H H H L	———
TRI-STATE AMPLIFIER LOW ENABLE			L L L L H H H L Z H H Z	———
TRI-STATE INVERTER LOW ENABLE			L L H L H L H L Z H H Z	———



TRUTH TABLE

INPUTS				OUTPUTS	
PRESET	CLEAR	CLOCK	D	Q	Q'
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H	H
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	HLD	HLD
H	H	H	X	HLD	HLD

TYPICAL OPERATION

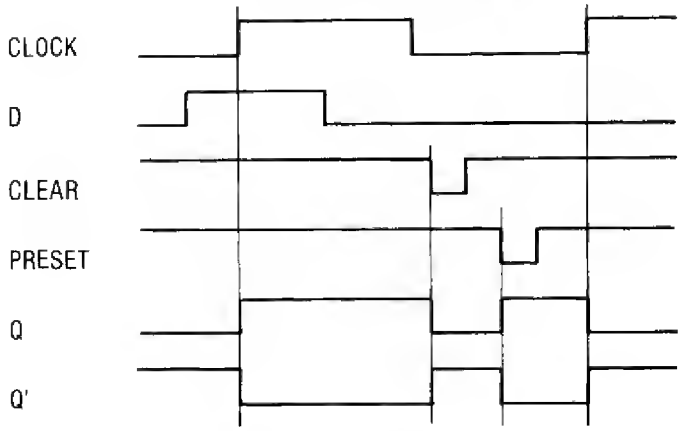


Figure E.1 A D-Type Flip Flop.

and the prime symbol represents the NOT or INVERSION function.* This selection of symbols makes the equation look like an equation of common algebra, and it's no accidental coincidence. The manipulation of such equations has parallels in the field of algebra and is, in fact, referred to as Boolean algebra, after the symbolic logic pioneer, George Boole.

Other functions besides AND, OR, NOT, and parenthesis grouping can be represented in logic equations, but only these basic functions are represented in logic equations in this book. The purpose in using such equations in *Understanding the Apple IIe* is only to describe some details of signal generation in a concise way. No algebraic manipulations are described, and none are required on the part of the reader.

*In representing the NOT function with a prime symbol, this book is following the sensible lead of the *Apple II Reference Manual for IIe Only*. The more common convention is to overscore the term or terms to which the NOT function is applied. The overscore is not a particularly workable representation because it is not a common typographical symbol and, more importantly, there is no code for it in standard computer text coding systems such as ASCII. Engineering and manufacturing printouts normally use an asterisk or prime symbol after a term to which the NOT function is to be applied. Apple should be commended for taking the lead in using this notation in published documents.

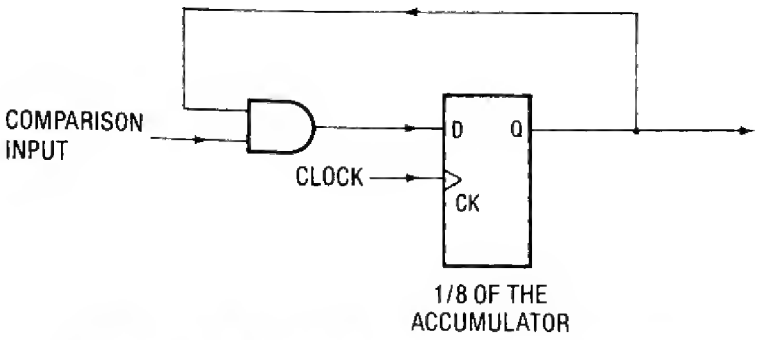


Figure E.2 The Circuit Equivalent of the 6502 AND Instruction.

appendix F

A Number System Primer

In our daily lives, we represent numerical quantities in the base 10, or decimal, numbering system. For example, by 359 we mean the sum of 9 x (10 EXP 0) plus 5 x (10 EXP 1) plus 3 x (10 EXP 2). This use of the decimal numbering system gives us some unusual biases that would occur only to mathematicians if we used a different base for our numbering systems. For instance, we place special significance on numbers like 1,000,000 (10 EXP 6) but not on 2,985,984 (12 EXP 6, equal to 1,000,000 in the base-12 or duodecimal numbering system). Mathematicians have studied number systems for years, but now, because of the growing influence of computers, knowledge of numbering systems other than base 10 is becoming very common indeed.

You see, the electronics of digital computers is based on hundreds of thousands of 2-state, or binary, electronic switches which can be on or off. The on or off state of each binary switch can be represented numerically as a ONE or a ZERO, and the information as to whether the switch is on or off is a bit of information. The simultaneous states of eight binary switches can be combined into an 8-bit binary word such as 10011110. Because of the 2-state nature of digital computer building blocks, digital analysis and design has been performed since day one using the base 2, or binary, numbering system. In this system, there are two digits—1 and 0. The binary number 110 represents the sum of 0 x (2 EXP 0) plus 1 x (2 EXP 1) plus 1 x (2 EXP 2), which is equal to 6 in decimal.

Actual performance of binary arithmetic is very unwieldy, particularly if you consider fractions. Addition and subtraction of 6502 addresses would require 16 digits. For example, subtracting decimal address 35000 from 35003 looks like this:

```
  1000100010111011
- 1000100010111000
                 11
```

If that looks clumsy, you should try multiplying 143 x 247:

```
      11110111
   x  10001111
      --------
      11110111
     11110111
    11110111
   11110111
 11110111000
--------------
1000100111111001
```

To prevent carrying out operations like this, computer programmers use other number systems based on powers of two, such as octal (base 8) and hexadecimal (base 16). Arithmetic is much easier to perform in these systems and conversion to and from binary is so easy that you can do it on sight with a little practice. For example, the above product can be read as hexadecimal 89F9 or octal 104771.

Conversion between binary and octal consists of dividing the binary number into groups of three, from right to left:

1 000 100 111 111 001 = 104771 (base 8)

These patterns of three digits can each be converted to one of eight octal digits from Table F.1. With exposure, these patterns become very familiar. As you would guess, there are eight symbols in the octal system, 0—7.

The hexadecimal system has 16 digits, 0—9, A, B, C, D, E, and F. The use of letters to represent numbers is sometimes confusing, but that's the convention we're stuck with. When converting between binary and hexadecimal, the binary number is divided into groups of four digits starting from the right:

1000 1001 1111 1001 = 89F9 (base 16).

6502 programming convention calls for use of the hexadecimal numbering system for representing addresses, machine language code, and much data. Convention further calls for preceding hexadecimal numbers with a dollar sign ($89F9) and binary numbers with a percent sign (%10001001), to distinguish them from decimal numbers. Following convention, the Apple monitor represents all numbers in hexadecimal. As a result, some skill in hexadecimal arithmetic and hexadecimal/decimal conversion is very desirable for Apple programmers. In addition, the well rounded computer programmer will be familiar with the binary and octal systems.

Here are two numerical facts of life about 6502 based microcomputers like the Apple. First, there are 16 address lines connected to the 6502. 16 lines can be in 65536 different possible combinations of states (0—65535, $0—$FFFF, or %0—%1111 1111 1111 1111). Second, there are eight data lines connected to the 6502. Eight lines can be in 256 different possible combinations of states (0—255, $0—$FF, or %0—%11111111). These numerical features of the 6502 account for some limiting numbers which occur in the BASIC language like 65535, 255, 32767, and 127.

Addresses are normally referred to in hexadecimal in *Understanding the Apple IIe*. This is because the hexadecimal representations make sense and

Table F.1 Number System Equivalent Representations.

DECIMAL	BINARY	HEXADECIMAL	OCTAL
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

are easy to remember. Numbers like $10000, $C000, and $2000—$3FFF are much more to the point than 65536, 49152, and 8192—16383 when representing the number of Apple addresses, the start of I/O addressing, and the address range of the HIRES PAGE 1 memory map. It is also far easier to remember that the cassette output port can be toggled by a reference to any address in the range $C02X ($C020—$C02F) than the range 49184—49199. Dear reader, the effort you spend learning your binary and hex systems will be well worth it.

If this is your first exposure to number systems, then you have just scratched the surface. It is highly recommended that you spend some time with mathematics or computer arithmetic courses, solving problems and familiarizing yourself with this sort of numerical manipulation. You need to get a little used to thinking like a computer. Probably the best exercise possible for you would be to write some 6502 assembly language programs. You'll sink or swim in hexadecimal and binary number systems.

appendix G

Revisional Information

The Apple IIe computer was released in 1983 with its 820-0064-A (Revision A) motherboard. This event had been well anticipated, but there was a little surprise for new purchasers of the Apple IIe and 64K RAM card. The RAM card manual described 560-point HIRES graphics that were only available on a Revision B motherboard with a 64 RAM card installed in the auxiliary slot. Revision B? Improved capabilities? Available soon? Why do they do these things to me? Why do they do these things to themselves?

Revision B was released a few months later, after all the Revision A inventory had been cleared. Relatively few Revision A Apple IIe's were sold and most of those that were sold were traded in for or converted to Revision B Apple IIe's. There has been no operational change to the motherboard since Revision B, so Revision B effectively is the Apple IIe.

The primary operational improvement of Revision B is the addition of the DOUBLE-RES graphics capability. This was achieved by rewiring 1/3 of the C5 LS10 NAND gate so that pulling pin 55 of the auxiliary slot low forces TEXT processing at the timing HAL instead of disabling motherboard ROM as it did in Revision A (see Figures 3.9 and 6.1). The nomenclature of pin 55 was appropriately changed from ENFIRM to FRCTXT'.

Changes in the timing HAL and MMU were required to make the new FRCTXT' wiring work. The Revision B timing HAL treats pin 12 as GR+2' whereas the Revision A timing HAL treats pin 12 as GR+2. The MMU was changed so that ROMEN1' and ROMEN2' are gated by INHIBIT', a necessity since the ROM inhibiting function had been performed by the NAND gate which now performs the forced text function.

It is my understanding that the video timing relationships were different in the Revision A timing HAL from what they are in Revision B. I have never examined the signal outputs of a Revision A HAL, so I'm not positive what the differences are. I have noticed that the VID7M/LDPS' relationship depicted on page 159 of the *Apple II Reference*

Manual for IIe Only is incorrect for the Revision B HAL, and I speculate that it may show the Revision A timing. In any case, wiring and component changes were made to the video summing amplifier in Revision B, and these would be necessary to realign COLOR REFERENCE to the PICTURE signal if video timing in the HAL were changed.

Another area of change in Revision B was in the functions of motherboard configuration jumper pads. Two new jumpers were added (X3 and X7), and the functions of X1 and X2 were changed. The functions of all Revision A and B jumper pads as well as the functions of the jumper pads on the PAL motherboard are given below.

The revisional history of the PAL motherboard is very similar to that of the NTSC motherboard. Its Revision A version was released (with no DOUBLE-RES graphics) at the same time as the Revision A NTSC motherboard. The Revision B version with DOUBLE-RES graphics was released soon afterwards, and Revision B is still current. An additional change that was incorporated in Revision B to the PAL motherboard was the inclusion of the color killing switch in the production motherboard. This late addition did not get into the Revision A production board, but it was added to the Revision A boards in a separate production step.

In summary, the characteristics that truly represent the Apple IIe are those of the NTSC and PAL Revision B motherboards, and the primary change of Revision B was the addition of DOUBLE-RES graphics modes. Differences between the NTSC and PAL motherboards were noted in Chapters 3 and 8. Essentially, the PAL motherboard is identical to the NTSC motherboard with the exceptions of PAL video generation, alternate language character set selection, 14M frequency, and IOU vertical scan rate.

APPLE IIe JUMPER PAD FUNCTIONS

Most Apple IIe owners never need to solder or break any of the motherboard or keyboard configuration jumper pads. On the other hand, readers of this book are just the type of people who would go in there and reconfigure the computer. Descriptions of the jumper pad functions follow here.

X1, X2—Character Set Switching

(NTSC Revs A and B, Figures 7.4, 8.5, 8.6)

Strictly for the purpose of character set switching, there really is no reason why there should be any jumper pads in this area, and there are none on the PAL motherboard. The PAL motherboard contains two language sets in the keyboard ROM and in the video ROM, and a mechanical switch connected to J19 will switch between video and keyboard sets simultaneously via the ALTCHR line.

The keyboard ROM on the NTSC motherboard contains two character sets, but the video ROM contains only one. In Revisions A and B, the normal configuration is for only the standard keyboard set to be enabled. This configuration can be changed via X1 and X2.

In NTSC Revision A, X2 is normally open, and the keyboard ROM alternate set selection line is normally connected to ground through X1. By opening X1 and closing X2, the owner can remove the ground and connect AN2 to the keyboard ROM alternate set selection line. This enables programmable switching of keyboard sets via AN2. To connect a mechanical switch, leave X2 open and open X1. J19 does not exist in NTSC Revision A, but the select wire from the switch can be soldered to X1 or X2 or to pin 19 of a separate socket installed between the keyboard ROM and its motherboard socket.

In NTSC Revision B, the keyboard ROM alternate set selection line is connected through the normally closed X2 jumper pad to the ENVID' line and through the normally open X1 jumper pad to ground. In this configuration, the alternate keyboard set is selected any time the video ROM is disabled via ENVID' high. To leave the standard set selected when ENVID' is high, open X1 and close X2. To select the alternate set via a mechanical switch without disabling the video ROM, open X2 and leave X1 open. As with Revision A, the switch can be soldered to X1 or X2 or to pin 19 of a separate socket installed between the keyboard ROM and its motherboard socket.

X3—AN2 Control of ENVID'

(NTSC Rev B, Figure 8.5)

In all versions of the Apple IIe, pulling the ENVID' line high disables the motherboard video ROM. One would only expect this to be done in conjunction with an auxiliary card that substituted an alternate picture signal on the ALTVID' line. An auxiliary card can pull ENVID' high itself, but there are two other options in the Revision B NTSC Apple IIe. The owner can install a jack and switch at J19 to give mechanical switch control over ENVID', or the owner can solder the X3 jumper so that programs can control ENVID' via AN2. As noted above, the X2 jumper should be

opened if it is desired that ENVID' be brought high without selecting the alternate keyboard set.

X4, X5—Performing DMA Without Stopping the MPU Clock

(All Apple IIe's, Figure 4.2)

With X4 and X5 normally configured, the PHASE 0 clock of the 6502 is prevented from rising when the DMA' line is held low by a peripheral card. PHASE 0 continues to alternate throughout the rest of the motherboard and peripheral slots, but it stays low at the MPU. This is fine for DMA peripheral cards which only need to steal a cycle or two of bus access from the 6502, but it means that peripherals which steal upwards of ten cycles in a row may cause the 6502 data registers to lose their values.

The owner can change this situation by opening X5 and closing X4. PHASE 0 will then continue to alternate at the 6502, even when DMA' is low. Care, however, must be taken by peripheral card designers wishing to use this feature because the 6502 will attempt to execute programs while not controlling the bus during DMA unless it is put into a wait state by pulling the READY line low. To perform long duration continuous DMA, pull READY low in conjunction with DMA'. To force feed the 6502 a program by addressing one area of memory while the 6502 is addressing another, pull DMA' low but leave READY high.

X6—Shift Key Mod

(All Apple IIe, Figure 7.2)

The SHIFT key mod is an improvised way of inputting upper and lower case characters on the upper case only Apple II. It consists of connecting a wire jumper between the SHFT' line and the PB2 game I/O input. PB2 then reacts to the SHIFT key, and text handling programs can read the keyboard ASCII and interpret it as upper or lower case depending on PB2. The SHIFT key mod is built into the Apple IIe through the X6 jumper pad. Just solder X6, and you will connect SHFT' to PB2, effectively installing the SHIFT key mod.

The SHIFT key mod is not a necessity on the Apple IIe unless you want to run an old Apple II program that works only with the SHIFT key mod and does not interpret direct upper and lower case ASCII from the keyboard. It is also advisable to solder X6 if you have no device connected that makes use of PB2 so you will have a convenient means of exercising PB2 if a program requires it.

X7—GR+2 Connection to Slot 7, Pin 23

(NTSC Rev B, PAL Rev B, Figure 7.6)

Pin 23 of Slot 7 was not connected to anything in Revision A. In NTSC and PAL Revision B motherboards, it is connected to GR+2 from pin 2 of the IOU through normally open jumper pad X7. The installation instructions of Slot 7 peripheral cards that require GRAPHICS/TEXT time identification will tell you to solder X7.

Keyboard CONTROL Required for Reset Jumpers

(All Apple IIe, Figure 7.4)

You must normally press CONTROL and RESET simultaneously to reset the Apple IIe. This is a design precaution that prevents accidental resets by mutinous or otherwise uncontrollable fingers. The precaution is not really necessary since the RESET key is recessed and separated from the rest of the keys (solitary confinement for crimes committed in the late 1970s), and it is inoperable for many handicapped persons. To change this so the RESET key generates a reset without the necessity of pressing CONTROL, remove the keyboard to gain access to its two unmarked jumper pads. Close the normally open pad and open the normally closed pad to make CONTROL not required for reset.

appendix H

Historical Notes

The original Apple was designed in late 1975 by Steve Wozniak, a talented college dropout who designed computers for fun. At some point in time, Wozniak entered into a partnership with his friend Steve Jobs, named the machine after a fruit, and sold a few hundred of these Apples. This original Apple had a 6502 microprocessor, 8K of RAM, no motherboard ROM beyond the screen text ROM, a motherboard power supply, and a single slot into which a cassette interface board plugged. The Apple was sold only as a circuit card, but enclosures and keyboards were available.

Of central importance to the first Apple was the 6502 microprocessor, which was then brand new. The 6502 was simple, powerful, and available for $20.00 over the counter to all comers. This accessibility made it an inviting MPU for an independent designer like Wozniak. Steve was a pioneer in building hardware around the 6502 and in programming the 6502. His BASIC interpreter was probably the first BASIC written for the 6502. This program was written directly in machine code, as were the system monitor and Wozniak's other early programs for the Apple.

In fall of 1976, Wozniak completed the design of the Apple II. This new computer far surpassed its predecessor in sophistication with HIRES and LORES graphics capability, 48K of RAM, BASIC and system monitor in ROM, built-in cassette I/O, and eight peripheral expansion slots with motherboard decoded slot control signals. The Apple II, no doubt, borrowed many features of hardware and program structure from the 1975 Apple, but most people would not recognize the older computer as an Apple.

While developing his Apple designs, Wozniak was not a lone talent working in solitude at his cerebral pastime. He was a member of the Homebrew Computer Club, the club to end all clubs, from whose membership rolls have come several microcomputer industry leaders. His friends were very interested in Steve's Apple and made substantial contributions to the Apple. Steve gives Allen Baum much of the credit for the peripheral

slot structure. In his "Apple II: System Description,"* he mentions Baum for originating the Apple II debug software, Doug Kraul for helpful suggestions on I/O structure, and Randy Wigginton and Chris Espinosa for testing Apple BASIC.

The contributions of Steve Jobs to the Apple II were of a different nature. Jobs was not very interested in designing computers, but he was very interested in selling the Apple. It was Jobs who thought big, who thought the Apple could be sold, and who pushed Wozniak in his development of a computer which was getting better and better. It was Jobs who talked big to people who counted: to Rod Holt who came over from Atari to help with electronic engineering tasks such as power supply design; to suppliers who were giving them components at discount prices with 30 days credit; to Mike Markkula who gave the new company business leadership and a quarter of a million greenbacks in seed money.

The Apple computer company officially came into existence in January of 1977. Company leaders included Markkula, Jobs, Wozniak, Holt, and Mike Scott who came over from National Semiconductor to be company president. Wozniak has a recollection of Scott answering phone calls to Apple while dubbing cassette tapes on a string of tape recorders. The company shipped its first Apple II in June of 1977 and had paid off all its debts by December of the same year. Growth of Apple II sales never stopped increasing as long as it was manufactured.

While the original Apple II was produced by a small group of talented and lucky individuals, more recent Apple products, including the Apple IIe, were produced by corporate Apple. Ideas for future products ebb and flow in a corporate entity for years, and so it was with the concept of an improved Apple II, more powerful, but cheaper to manufacture. Steve Wozniak was the force behind the original idea. He worked with Synertek on an early project (Apple code name Annie, circa 1978) that used custom ICs to perform many Apple functions. Annie was never developed into a final product, and I know nothing of its architecture or capabilities.

At some point after Annie was shelved, Apple engineer Burrell Smith began work on a different improved Apple II. Smith built an Apple II (code name Diana) in which many logic functions were performed by PALs. Like Annie, Diana never made it to production. Its development was sidetracked by Smith's assignment to the Macintosh project and by the fact that a custom IC approach to an improved Apple II seemed more promising. However, some of Diana's architecture, including the timing HAL and video ROM, was borrowed by the Apple IIe. In my opinion, the video ROM is the cleanest improvement in the design of the Apple IIe over that of the Apple II. I'm not sure who originally came up with the idea, but it may have been Smith or longtime Apple engineer Wendel Sander.

The engineer in charge of project Annie at Synertek was Walt Broedner. Walt left Synertek in 1981 and went to work for Apple, first on the Apple III, and later on the Apple II. At Apple Broedner continued to support the concept of using custom ICs as the basis for improving the Apple II design. To bring his point home, he designed the the MMU and the IOU, reworked the Diana video timing to make it support DOUBLE-RES graphics, and built a working prototype of the Apple IIe. In doing this, Broedner also designed the Apple IIc, because his prototype operated like an Apple IIc if you threw some switches.

It is certain that others besides Walt Broedner* had a hand in Apple IIe development. Engineering manager Peter Quinn oversaw the entire project. Gary Baker incorporated his Eurocolor card into the PAL motherboard and designed the video amplifiers of the NTSC and PAL motherboards. The new Apple IIe firmware was written by Rick Auricchio. As was mentioned previously, Burrell Smith's Diana work was significant. Wozniak, of course, was most influential since it was his computer that was being redesigned.

*_BYTE_ Magazine, May 1977

*After designing the Apple IIe and IIc, Walt Broedner left Apple to form a new company, Video-7 Inc.

appendix I

Apple II/IIe Difference Notes

Most Apple users are aware of the primary operational differences between the Apple II and the Apple IIe, and in fact, these differences have been well described in various writings. However, the magnification power of *Understanding the Apple IIe* is turned up a little higher than most Apple IIe writings, and examination at this level turns up differences too obscure to be noted in more general descriptions. So, for those who like the view of the trees from inside the forest, here are some notes on differences between the Apple II and IIe. These differences are listed under the chapter in which their related area is covered so the reader may know where to look for further explanations.

Chapter 1—Overview

As noted in Chapter 1, the Apple IIe is operationally compatible with a 48K Apple II with 16K RAM card in Slot 0, 80-column card in Slot 3, and an enhanced keyboard. The major improvements beyond this consist of motherboard support for 64K of auxiliary RAM (an auxiliary 48K plus an auxiliary Slot 0 RAM card) and the DOUBLE-RES graphics modes. Additionally, there are numerous minor operational differences between the two computers pointed out under Chapters 2—10 below.

The difference in hardware implementation between the Apple II and the Apple IIe are greater than the operational differences. Basically, the Apple II motherboard was rebuilt from the ground up using custom ICs, the timing HAL, bigger RAM and ROM chips, and redesigned logic structures to achieve and improve upon old features in simpler ways. In addition to motherboard hardware changes, a few changes were made to the case and base plate structures including a redesign of the back structure so that it is less convenient to string peripheral slot cables out the back, but so that the outside world is better protected from Apple RFI emission.

Chapter 2—Bus Structure

Distribution of the address bus and data bus in the Apple II and Apple IIe is, of course, similar. This is dictated by the nearly identical MPU, peripheral slot pin assignments, and operational features of the two machines. There are, however, numerous differences in the bus distribution details, most notably in the RAM area. The main reason for these differences is that packing the

logic functions of many small ICs into the two large custom ICs required things to be done in new and different ways. Also, the larger RAM and ROM chips and auxiliary slot of the Apple IIe result in some differences.

1. **RAM Address Multiplexing**—In the Apple II, the address bus and video scanner provide inputs to a 4 to 1 multiplexor which outputs VIDROW, VIDCOL, MPUROW, and MPUCOL addresses in turn to RAM on seven lines. The 4 to 1 multiplexor does not have tri-state outputs, so the multiplexed RAM address bus never floats. In the Apple IIe, the MMU and IOU drive the 8-line RAM address bus with tri-state outputs. While the MMU presents a high impedance, the IOU places the VIDROW then VIDCOL addresses on the RAM address bus. While the IOU presents a high impedance, the MMU places the MPUROW then MPUCOL addresses on the RAM address bus. The MPUROW address is input to the IOU as well as to RAM. The RAM address bus is distributed to auxiliary card RAM in addition to motherboard RAM, and there is a short period of time just before the VIDROW and MPUROW addresses when the RAM address bus floats.
2. **RAM Output Data Distribution**—In the Apple II, RAM output data is latched and distributed to the RAM/keyboard data multiplexor and to the video generator. The RAM/keyboard data multiplexor places either the latched keyboard data or the latched video data on the data bus when a read is made in the $0000—$C00F range. In the Apple IIe, motherboard RAM output data goes directly to the data bus for reading by the MPU and saving in the motherboard video latch. Auxiliary RAM data input and output lines are connected to the auxiliary video latch and to a bidirectional driver. The bidirectional driver passes data between auxiliary RAM and the data bus when the MPU is reading from or writing to auxiliary RAM. The auxiliary video latch saves video data from auxiliary RAM and gates the video data to the video data bus at the correct time for processing in the video generator. The keyboard circuits in the Apple IIe have their own tri-state connection to the data bus.
3. **The Bidirectional Bus Driver**—In the Apple II, there is a bidirectional driver between the MPU and the data bus. This enables the MPU to drive the data bus during write cycles, even with all the motherboard and peripheral card devices connected to the data bus. In the Apple IIe, the bidirectional driver is situated between the peripheral slots and the data bus. The MPU must therefore drive the heavy motherboard data bus load during write cycles without the aid of a driver. This load is constant and does not vary with peripheral card installation. An advantage of the Apple IIe implementation is that motherboard devices like RAM and ROM have more driving power when supplying data to the peripheral cards during DMA read cycles. Also, the Apple IIe design tolerates more peripheral card data bus loading than that of the Apple II.

Chapter 3—Timing Generation and the Video Scanner

The primary timing signals of the Apple II and the Apple IIe—14M, 7M, COLOR REFERENCE, RAS', AX, CAS', Q3, PHASE 0, and PHASE 1—are nearly identical. The frequencies, sequences, and the long cycle are the same in the two computers. The video scanner is also functionally identical in the Apple II and IIe with 65 cycles per horizontal scan, 262 horizontal scans per vertical scan in the 60 Hz Apple, and 312 horizontal scans per vertical scan in the 50 Hz Apple.

Some notable differences:

1. CAS' is gated by CASEN' from the MMU during PHASE 0 in the Apple IIe. This is the way motherboard RAM is enabled or disabled. In the Apple II, CAS' always falls after AX falls, and CAS' from the timing generator is gated to one of three octets of RAM chips during PHASE 0 by $0000—$3FFF, $4000—$7FFF, or $8000—$BFFF addressing.
2. CAS' in the Apple II rises simultaneously with RAS'. RAS' rising latches RAM read data, and CAS' rising causes the RAM chips to bring their data outputs to high impedance after a short delay. In the Apple IIe, RAM read data is not latched by RAS' rising so CAS' rising had to be moved back closer to the end of the 6502 cycle. CAS' rises one 14M period after RAS' rises (simultaneously with PHASE 0 or PHASE 1 rising).
3. In the Apple II, the long cycle occurs at a horizontal scan count of 0000000. In the Apple IIe, the long cycle occurs 1 $^1/_2$ scan counts later, straddling horizontal states 1000000 and

1000001. The reason for the difference is that there is a greater lag between RAM addressing and video output in the Apple IIe timing generator. In the IIe, the long cycle had to be pushed back so it would not interfere with the last video output cycle of each horizontal scan.

4. An Apple IIe auxiliary card can monitor the timing signals and substitute its own signals for the motherboard timing signals. There is no similar capability in the Apple II.
5. The flash counter, which is an extension of the video scanner, does not exist in the Apple II. The Apple II uses analog timers to produce time references for flashing text, the keyboard repeat function, and the power-up reset.
6. The video timing signals, LDPS' and LD194 in the Apple II and LDPS' and VID7M in the Apple IIe, are entirely different. These differences are due to the differences in the video generator hardware and the 80-column display capability of the Apple IIe.
7. The hardware implementation of the timing generator and video scanner in the Apple IIe is different from that of the Apple II, but the difference is mainly in the use of state-of-the-art ICs to achieve the same functions. (There is still an underlying similarity in the hardware.) In both machines, the timing generator is made up of a 14.31818 or 14.25045 MHz oscillator whose output is divided and processed to make up the same timing signals. Also in both machines, the video scanner is a 262/65 or 312/65 state counter whose outputs address display memory and trigger events related to television scanning.

Chapter 4—The MPU

Both the Apple II and the Apple IIe use 6502 microprocessors, and most 6502 related information in the two computers is identical. This includes 6502 programming, speed of program execution, interrupts, and the control signal connections between the 6502 and the peripheral slots. However, there are numerous minor differences in 6502 connections and related circuits. Some of these differences are listed here.

1. The Apple II uses the 1 MHz 6502 while the Apple IIe uses the 2 MHz 6502A. The Apple IIe MPU is therefore tested to operate with shorter delays between events.
2. The Apple II 6502 is connected to the data bus through an external bidirectional bus driver whose direction is controlled by R/W' and 6502 PHASE 1. The Apple IIe 6502 is connected directly to the data bus. 6502 PHASE 1 in the Apple IIe is not connected.
3. The 6502 SYNC signal is connected to pin 39 of the peripheral slots in the Apple IIe. 6502 SYNC is not connected in the Apple II.
4. The 6502 SET OVERFLOW' input is grounded in the Apple II, but it is open in the Apple IIe.
5. Wire-OR control inputs from the peripheral slots are pulled up with 1000 ohm resistors in the Apple II and 3300 ohm resistors in the Apple IIe. This results in slower low to high switching speed in the Apple IIe which can be offset, when necessary, with parallel pull-up resistors on peripheral cards.
6. The X4 and X5 jumpers of the Apple IIe allow peripheral slot DMA operations which do not stop the PHASE 0 clock to the 6502. In an unmodified Apple II, pulling DMA' low always inhibits the 6502 PHASE 0 clock.
7. In the Apple IIe, CAS' is pre-gated (CASEN' gating occurs prior to 14M clocking), and CAS' is applied directly to the RAM chips. In the Apple II, CAS' is post-gated (RAM SELECT' gating occurs after 14M clocking) before application to the RAM chips. As a result, in relation to PHASE 0 and 6502 PHASE 2, CAS' in the Apple IIe falls typically 21 nanoseconds (32 nsec max) before CAS' falls at the Apple II RAM chips. 6502 write data must, therefore, be set up earlier in the Apple IIe than in the Apple II. This is one good reason for the use of a 2 MHz 6502A (100 nsec write data setup) instead of a 1 MHz 6502 (175 nsec write data setup) in the Apple IIe.
8. Because of long address bus to multiplexed RAM address bus propagation delay in the MMU, DMA peripherals must set up the address bus earlier in the Apple IIe than in the Apple II.
9. NMI', IRQ', and BREAK handlers in the Apple II with Autostart Monitor and the Apple IIe are identical. Reset operations are similar in most aspects, including disk autostart, the power-up byte, and the RAM reset vector. The Apple IIe reset handler also supports open Apple and solid Apple resets, and its video initialization routines support the improved Apple IIe video features.
10. The Apple IIe can operate with a 65C02 microprocessor but an Apple II cannot. I think that this is because RAM read data in an Apple II is set up too late for the 65C02. I have verified that

NCR 65C02s do not work reliably in the Apple II, but I have not verified this for Rockwell 65C02s.

Chapter 5—RAM and Memory Management

No area of the Apple II and IIe computers is more different in hardware implementation than RAM and memory management. This is in spite of the fact that the Apple IIe is operationally similar to a 48K Apple II with a 16K RAM card installed in Slot 0. Differences center around the use of 64K RAM chips in the Apple IIe and the more versatile memory management features of the Apple IIe.

1. The Apple II has 48K of motherboard RAM in 24 16-kilobit chips. The Apple IIe has 64K of motherboard RAM in eight 64-kilobit chips.
2. The 16K RAM card is not affected by RESET' falling, is not inhibited by the INHIBIT' line, does not support reading of the configuration soft switches, and steals $F800—$FFFF address response for its F8 ROM when its RAM is disabled for reading. Apple IIe high RAM is disabled for reading and enabled for writing when RESET' falls and is inhibited when INHIBIT' is low. The state of the HRAMREAD and BANK1 soft switches can be read at $C012 and $C011.
3. Apple IIe memory management fully supports access to 64K of auxiliary card RAM. There is no equivalent capability in the Apple II.
4. The RAM address multiplexor of the Apple II is a 4 to 1 multiplexor made up of a number of TTL chips driving seven 2-state RAM address lines. The RAM address multiplexor of the Apple IIe is a 4 to 1 multiplexor made up of the 2 to 1 video multiplexor in the IOU and the 2 to 1 MPU multiplexor in the MMU which together drive eight tri-state RAM address lines. The overall multiplexing functions are very similar including identical correspondence between displayed video address and MPU address in the two computers.
5. HBL scanned addresses in TEXT/LORES scanning in the Apple II are $1000 higher than HBL' scanned addresses. This effect of HBL resulted from the Apple II scheme of refreshing memory in a computer that would accept 16K or 4K RAM chips. HBL gating is not included in the Apple IIe RAM addressing so HBL scanned addresses overlap the HBL' scanned addresses in the Apple IIe.
6. Since HBL is not a RAM address input in the Apple IIe, SUM A4 and SUM A5 are adequate refresh signals and VC does not have to be used for RAM refresh as it is in the Apple II. As a result, the refresh period in the Apple IIe never exceeds 2 milliseconds. In the Apple II, the refresh period slightly exceeds 2 milliseconds in HIRES mode during a portion of VBL.
7. Motherboard RAM is connected to the data bus through buffering devices in the Apple II but is connected directly to the data bus in the Apple IIe. This affects data bus management in all MPU communication in that you don't have the RAM/keyboard data multiplexor jumping on the Apple IIe data bus right after PHASE 1 goes high.
8. Details of MPU/RAM communication timing are different in the two computers and generally less critical in the Apple IIe. A notable exception is that the 6502 has about 21 nanoseconds less time to set up RAM write data in the Apple IIe than it does in the Apple II (see Chapter 4, item 7 above). Also, because of long address bus to multiplexed RAM address bus propagation delay in the MMU, DMA peripherals must set up the address bus earlier in the Apple IIe than in the Apple II.
9. The MMU and most of its capabilities do not exist in the Apple II. Of the MMU soft switches, 80STORE, RAMRD, RAMWRT, INTCXROM, SLOTC3ROM, INTC8ROM, and ALTZP and the memory management functions of PAGE2 and HIRES do not exist in the Apple II. Additionally, HRAMRD, BANK1, PRE-WRITE, and HRAMWRT' exist only on a 16K RAM card in an Apple II, not on the motherboard. MMU functions that do not exist on the motherboard of an Apple II include auxiliary RAM management, high RAM/ROM management, $C100—$CFFF switching between I/O and ROM, inhibiting of RAM via INHIBIT', MPU reading of soft switch states, disabling of high RAM at RESET', and management of a peripheral data bus driver. MMU functions that do exist in the Apple II include RAM address multiplexing and address decoding of RAM, ROM, I/O, and keyboard enabling signals.
10. The capability of resetting the MMU from a program that was noted in Chapters 4 and 5 applies only to the Apple IIe. There is no equivalent capability in the Apple II.

Chapter 6—ROM

The function of ROM is so basic and the contents of Apple II and IIe firmware are so similar that not much can be said to have changed in this area. There are a few basic differences.

1. The Apple II has 12K of firmware residing in six 2K ROM chips compared to 16K of firmware residing in two 8K ROM chips in the Apple IIe. The nature of the extra firmware is noted in Chapter 6. Basically, it supports the 80-column, upper/lower case capabilities and other extended features of the Apple IIe. Notable firmware improvements include extra control and escape sequences when 80-column firmware is active and the optional forced boot reset (open Apple reset).
2. Motherboard ROM response to $C100—$CFFF addressing is not a capability of the Apple II.
3. MPU/ROM communication timing details are very similiar although Apple IIe ROM timing lags Apple II ROM timing because of long MMU propagation delay.
4. ROM in the Apple II is not pin compatible with EPROM except for the text ROM of Revision 7 and later motherboards. All ROM on the Apple IIe motherboard is pin compatible with EPROM.
5. The Apple IIe firmware upgrade improvements noted in Chapters 4 and 6 are not available in the Apple II.

Chapter 7—I/O

The I/O features and structure of the Apple IIe are compatible with those of the Apple II to a very great extent. Nevertheless, there are 1001 itsy bitsy differences. Here are a few of the 1001.

IOU Soft Switches:

1. The IOU does not exist in the Apple II but most IOU capabilities and soft switches do.
2. The window for toggling a soft switch in the Apple II is a period equal in duration but slightly lagging PHASE 0. The window for toggling the equivalent IOU soft switches in the Apple IIe is a period slightly lagging $\Phi 0 \bullet Q3' \bullet RAS''$.
3. The ALTCHRSET, 80STORE, and 80COL soft switches do not exist in the Apple II. The Apple II only has modes equivalent to $ALTCHRSET' \bullet 80STORE' \bullet 80COL'$.
4. There is no equivalent to the capability of reading VBL' and AKD in the Apple II.
5. TEXT, MIXED, PAGE2, and HIRES exist in the Apple II, but their states cannot be read by the MPU as they can in the Apple IIe.
6. Read access to $C01X in the Apple II resets the keyboard strobe flip-flop. Read access to $C01X in the Apple IIe passes the state of AKD, VBL', or an MMU or IOU soft switch to MD7 of the data bus and passes keyboard ASCII to MD6—0 of the data bus. Read access to $C010 also resets the KEYSTROBE soft switch in the Apple IIe. Write access to $C01X resets the equivalent of the KEYSTROBE soft switch in either computer.
7. The KEYSTROBE auto repeat feature of the Apple IIe does not exist in the Apple II.
8. AN0—AN3, PAGE2, and HIRES are reset when RESET' falls in the Apple IIe but not in Apple II.
9. The timing of Apple IIe output signals originating in the IOU is generally similiar to, but slightly lags due to IOU propagation delay, the timing of same signals in the Apple II. This includes AN0—3, SPKR, and CSST OUT timing.

Serial I/O:

1. The 9-pin game I/O extension jack in the back of the Apple IIe does not exist in the Apple II.
2. PB0 and PB1 are tied to open Apple and close Apple and pulled low through 470 ohm resistors on the keyboard in the Apple IIe but not in the Apple II.
3. In the Apple IIe, but not the Apple II, the SHIFT key mod can be installed (PB2 connected to the SHFT' line) by soldering a motherboard jumper pad.
4. The open collector outputs of the quad timer have 1K pull-up resistors in the Apple IIe but not in the Apple II.
5. A self-test LED is connected across the speaker jack in the Apple IIe to give a firmware diagnostic pass indication when the motherboard is powered up with no keyboard or speaker connected. This LED is not present in the Apple II.

Keyboard:

1. Early Apple II keyboards cannot input ASCII for lower case alphabetic and some control characters. Later Apple II keyboards can input lower case alphabetic characters if the

user installs a switch, but the lower case keyboard input is not supported by Apple II firmware. The Apple IIe keyboard can input all 128 ASCII codes and there is some degree of firmware support for lower case keyboard input.

2. There are a number of keys on the Apple IIe keyboard that are not on the Apple II keyboard including DELETE, TAB, CAPS LOCK, open and close Apple, up and down arrow, and several special character keys.
3. The layout of Apple II special character keys is similar to that of a teletype. The layout of Apple IIe special character keys is similar to that of an IBM *Selectric* typewriter.
4. The Apple IIe keyboard ROM with dual keyboard layout enables the user to switch between keyboard layouts or reprogram the two layouts. There is no similar capability in the Apple II.
5. The Apple IIe motherboard contains a jack for a numeric keypad. The Apple II motherboard does not have such a jack although later Apple II keyboards have holes meant for installation of a keypad jack.
6. See also items 4, 6, and 7 under IOU Soft Switches and item 2 under Serial I/O above.

Peripheral Slots:

1. There are only seven peripheral slots (1—7) in the Apple IIe compared to eight peripheral slots (0—7) in the Apple II. The Apple II Slot 0 DEVICE SELECT' range, $C08X, is used to configure high memory in the Apple IIe.
2. The bidirectional data bus driver in the Apple II is situated between the MPU and other data bus devices. The bidirectional data bus driver in the Apple IIe is situated between the peripheral slots and serial input multiplexor and the other data bus devices.
3. An Apple IIe Slot 1 peripheral card can disable motherboard timing or the keyboard circuits via CLKEN' and ENKBD' respectively. These lines do not exist in the Apple II.
4. Pin 39 is connected to the USER1 line in the Apple II and the 6502 SYNC signal in the Apple IIe. Any peripheral card in the Apple II can disable all $CXXX I/O decoding by pulling USER1 low. A similar capability in the Apple IIe is that any program can disable $C100—$CFFF I/O decoding by manipulating INTCXROM, SLOTC3ROM, and INTC8ROM. $C0XX I/O decoding cannot be disabled in the Apple IIe.
5. The INHIBIT' line of the Apple II inhibits motherboard ROM but does not affect RAM. The INHIBIT' line of the Apple IIe inhibits motherboard and auxiliary card memory (RAM and ROM).
6. Pins 19 of Slots 0—6 and pins 35 of Slots 0—6 are connected together but not connected to any signal in the Apple II. Pins 19 and 35 of Slots 2—6 in the Apple IIe are not connected.
7. Peripheral slot wire-OR lines are pulled up by 1K motherboard resistors in the Apple II and by 3.3K motherboard resistors in the Apple IIe.

Auxiliary Slot:

1. The auxiliary slot and the capabilities associated with it exist only in the Apple IIe, not in the Apple II.

Chapter 8—Video Generation

Apple IIe video output is compatible with Apple II video output to a very great extent. SYNC, COLOR BURST, and blanking logic equations in the Apple IIe are identical to those of the later Apple II (RFI Revision), and SINGLE-RES Apple IIe displays are identical to Apple II displays produced by the same memory map (exept as noted in item 7 below). The big operational change in video generation is the addition of the DOUBLE-RES modes. But the most striking change to a student of Apple II hardware is the cleanup of the video generator design.

The differences:

1. The double horizontal resolution display modes (80-character TEXT, 80-block LORES, and 560-point HIRES) of the Apple IIe do not exist in the Apple II. Neither do the high speed timing and auxiliary memory which are required to support the double resolution modes.
2. The design of the video generation circuitry is much cleaner in the Apple IIe than it is in the Apple II. In addition to integrating many video logic functions into the IOU, the text pattern ROM, text shift register, HIRES/LORES configurable graphics shift register, picture selection multiplexor, and synchronizing flip-flop in the Apple II are replaced by the video ROM and one shift register in the Apple IIe.
3. Apple IIe displayed text characters and graphics characteristics can be changed by replacing the video ROM. Displayed text characters can be changed by replacing the text ROM in Revision 7 and later Apple IIs.

4. A standard Apple II motherboard can be configured for PAL video output by making the 50 Hz jumpers, installing a 14.25 MHz crystal on the motherboard, and installing a "Eurocolor" PAL encoding card in Slot 7. With the Apple IIe, different NTSC and PAL motherboards are used depending on the television system of the country they are operated in.
5. The Apple IIe can display 96 text characters. The Apple II can display only 64 text characters although this can be increased to 96 by changing the text ROM of Revision 7 and later Apple IIs.
6. The Apple IIe can switch between Apple II compatible inverse and flashing video and full ASCII inverse via the ALTCHRSET soft switch. The Apple II can operate only in ALTCHRSET' mode.
7. In Apple II HIRES40 mode, delayed patterns at the far left extend undisplayed dots onto the screen, and delayed patterns at the far right can be cut off by following undisplayed patterns. In Apple IIe HIRES40 mode, delayed patterns at the far left extend the left hand blanking margin, and delayed patterns at the far right are always cut off.
8. The Apple IIe capabilities of monitoring and disabling video signals and injecting an alternate picture signal from an auxiliary card do not exist in the Apple II.
9. Low-high propagation delay is significantly shorter than high-low propagation delay on the PICTURE signal of the Apple II. This causes bright spots to be about 24 nanoseconds wider than equivalent dark spots. No such effect is noticeable in the Apple IIe.
10. At the signal sources, the PICTURE signal (referenced to COLOR REFERENCE) of the Apple II lags that of the Apple IIe by one 14M period. COLOR REFERENCE is delayed more in analog shaping circuits in the Apple II than the Apple IIe, so at the video summing amplifier, the PICTURE signal/COLOR REFERENCE relationship is identical in the two computers.
11. HIRES40 colors can be instantly switched between delayed and undelayed colors via FRCTXT' in the Apple IIe. There is no equivalent capability in the Apple II.
12. The abnormal LORES mode resulting from resetting 80COL and bring FRCTXT' low in the Apple IIe does not occur in the Apple II.
13. Video generation timing in the Apple II and IIe have similarities but are very different in detail. Apple IIe timing is delayed by video ROM access time, and DOUBLE-RES timing exists only in the Apple IIe.

Chapter 9—The Disk Controller

There is no difference in floppy disk I/O between the Apple II and Apple IIe. The controller, disk drive, and operating systems are not built into the Apple and therefore evolve separately from the computer. Advances made since the release of the Apple IIe include the development of several Disk II compatible drives, the introduction of ProDOS, and the development of the IWM (Integrated Woz Machine) which is a custom IC that emulates the Disk II controller and may be used in future Apple II/IIe controllers.

Chapter 10—Maintenance and Care

Much that can be said about maintenance and care of Apple computers is valid for either the Apple II or the Apple IIe. In the Apple IIe, peripheral slots, tinkering users, and the power supply remain as reliability weak links, and basic problem troubleshooting steps like removing peripheral cards and evaluating the video display are the same. Some things have changed, to be sure, and generally, for the better. Most notably the Apple IIe is more reliable, as has been amply verified by this world class tinkerer.

Apple IIe reliability improvements include stronger motherboard mounting near the peripheral slots and a reduced number of motherboard ICs. Additionally, the verification and fault isolation capabilities represented by the firmware diagnostics and the auxiliary slot are not present in the Apple II.

index

Understanding the Apple IIe

A Learning Guide and Hardware Manual for the Apple IIe Computer

"Understanding the Apple IIe leaves no stone unturned in the search into the inner workings of the Apple IIe computer."
—Steve Wozniak

Quality Software is pleased to present the definitive source of information about how the Apple IIe works. Jim Sather has followed up his exhaustive analysis of the inner workings of the Apple II computer with an even more detailed analysis of the Apple IIe. Now he has documented his findings in a way that will benefit everyone interested in microcomputer technology.

About the Author
James Fielding Sather, a former electronics field technical representative for ITT Gilfillan, is an independent author, programmer, and designer of circuits for microcomputers, specializing in the Apple II and Apple IIe.

Understanding the Apple IIe describes the operation of the Apple IIe computer. Its companion volume, ***Understanding the Apple II***, describes the operation of the Apple II and Apple II Plus.

***Understanding the Apple IIe*—**

- Documents all motherboard circuits, including diagrams and descriptions of the inner workings of the MMU, IOU, and timing HAL.
- Describes disk controller operation, including previously undocumented details of the logic state sequencer
- Reveals previously unnoticed features of Apple graphics. Explains double-res.
- Explains differences between the Apple II and Apple IIe.
- Contains 15 software and hardware Application Notes including EPROM mods, disk write protect mod, split screen programming, and the DOS HOSS firmware card mod
- Includes a chapter on maintenance that provides simple troubleshooting steps.
- Describes the PAL (European) motherboard video circuits.
- Contains more than 100 illustrations.
- Includes valuable programming reference material.
- Documents 1985 firmware upgrade.

If you are at all curious about how the Apple IIe wo you are sure to find ***Understanding the Apple IIe*** very valuable. It is an ideal book for a microcomput fundamentals course based on the Apple.

QS QUALITY SOFTWARE
Computer Book Division

Brady

PRINTED IN USA

ISBN 0-8359-8019-7