Cortland Firmware Preliminary Notes

Writer: Wayne Lowry
Apple User Education
030-1290-PN7

January 15, 1986

Changes Since Last Draft

- Cortland Monitor ERS 00.40, November 4, 1985
- Cortland Banks $E0/$E1 Memory Map ERS 00.40, Oct. 9, 1985
- Control Panel ERS 00.30, October 21, 1985
- Mouse ERS 00:10, July 15, 1985
- Columbia AppleTalk ERS 00.00, June 12, 1985
- Serial Ports ERS 00.10, November 5, 1985
- Disk Support ERS 00.10, November 6, 1985
- Front Desk Bus ERS 02.50, October 24, 1985
- Desk Accessory Manager Switcher ERS 00.00, June 17, 1985
- Memory Manager ERS Rev1, November 5, 1985
- Phoenix Memory Map ERS 00.00, July 15, 1985
- Phoenix Human Interface ERS 00.00, June 28, 1985

Acknowledgements

The Manual Team

Writer: Wayne Lowry
Editor: Dorothy Pearson
Writing Supervisor: Rolly Reed

The Product Team

Engineering: Fern Bachman, Mike Askins, G. Andrade, R. Montagne, Peter Baum

Contents

Preface

About This Manual

The Cortland firmware contains the programs or instructions that are stored in the system's read-only memories. These programs determine what functions the system can perform.

The Cortland firmware programs consist of The Monitor, Single-Chip Microcomputer Keyboard Interface (SKI), AppleTalk, The Control Panel, The Cortland Mouse, Disk Support System, and the Serial Ports.

Cortland Monitor

The system Monitor is a set of subroutines that provide a standard interface for the built-in I/O devices. The Cortland Monitor firmware also provides access to the rest of the system through standard entry points. Most of these functions are initiated by keyboard commands.

Control Panel

The Control Panel program is used for the configuration of hardware, and is displayed in 40 or 80 columns. It is invoked when the system is powered up or when called from an application program.

Cortland Mouse

The Cortland Mouse uses the Front Desk Bus (FDB) to communicate with the keyboard microprogram; the microprogram informs the monitor firmware of mouse activity.

AppleTalk

AppleTalk is a work area network which provides communications and resource sharing with up to 32 computers, disks, printers, modems, and other peripherals. AppleTalk consists of communications hardware and a set of communications protocols. This hardware/software package, together with the computers, cables and connectors, shared resource managers (servers), and specialized applications software function in three major configurations: small-area interconnect systems, a tributary to a larger network, and a peripheral bus between Apple computers and their dedicated peripheral devices.

Serial Ports

The Serial Ports perform serial communications for Cortland. The serial-port firmware supports all command codes used by the Super Serial Card (SSC), input buffering, and background printing.

Disk Support

The Disk Support System accommodates Sony 3.5-inch disk drives with or without built-in intelligence.

Single Chip Microcomputer Keyboard Interface (SKI)

The SKI is the interface between the system processor and the Single-chip Microcomputer. The Front Desk Bus (FDB) and the internal keyboard are controlled by SKI software protocols.

Cortland Banks $E0/$E1 Memory Map

Although Banks $E0 and $E1 are not firmware as such, they are the portion of RAM used by the Cortland firmware to perform its various functions.

NOTE: In addition to the above functional programs, the ROM also includes Tool routines and Diagnostics that are described in another document.

Cortland Tools

The Cortland tools provide a means of constructing application programs that conform to the standard user interface. By offering a common set of routines that every application can call to implement the user interface, the tools not only ensure familiarity and consistency for the user, but also help to reduce the application's code size and development time.

Diagnostic Routines

The diagnostic routines stored in ROM serve to test the system automatically when the system is powered up. Also, the option is available to invoke tests stored on disk for a more extensive system and peripheral test.

Chapter 2

Serial Ports

Introduction

The serial ports perform serial communications for Cortland. The serial-port firmware supports all command codes used by the Super Serial Card (SSC), input buffering, and background printing. (Background printing is accomplished in cooperation with the interrupt handler firmware.)

The Cortland serial ports use a two-channel Serial Communications Controller chip (SCC). (The Super Serial Card and Apple II use 6551s.) Serial port 2 shares its channel of the SCC chip with AppleTalk. AppleTalk and serial port 2 cannot be active at the same time in any shared or swapped mode (the Control Panel program ensures that the port 2 serial firmware is inactive when AppleTalk has been selected). Either port 1 or port 2 can be configured as a printer or a communications (modem) port.

In case of a conflict between Apple II control commands and SSC commands, the SCC is implemented.

Default parameters for the serial ports are set by the user in the Control Panel program. The application program can change the parameter values temporarily by sending control sequences to the serial-port firmware.

Compatibility

The commands used to communicate with the serial-port firmware are the same as those used with the Super Serial Card and Apple II ports; however, many existing programs using these ports will not be compatible with the Cortland serial ports. Many programs, particularly communications packages, go directly to the hardware; the hardware no longer uses a 6551. Print programs are more likely to work, as well as applications written in BASIC and Pascal. AppleWorks and MousePaint are examples of programs that are compatible with the firmware.

Control Commands

The firmware accepts commands in the following sequence:

<CommandChar> <CommandString> <Cr>

When a port is in the printer mode, CommandChar is a CONTROL-I, and in communication mode, it is a CONTROL-A. The CommandString is a letter command, sometimes prefixed by a number or suffixed with an E or D.

The following commands invoke both printer and communications modes:

<n>B — Set the baud rate to a value corresponding to n.

0B = (use default)	6B = 300 baud	12B = 4800 baud
1B = 50 baud	7B = 600 baud	13B = 7200 baud
2B = 75 baud	8B = 1200 baud	14B = 9600 baud
3B = 110 baud	9B = 1800 baud	15B = 19200 baud
4B = 134.5 baud	10B = 2400 baud	
5B = 150 baud	11B = 3600 baud	

<n>D — Set data format to values per n (data bits, stop bits).

0D = 8 data, 1 stop	4D = 8 data, 2 stop
1D = 7 data, 1 stop	5D = 7 data, 2 stop
2D = 6 data, 1 stop	6D = 6 data, 2 stop
3D = 5 data, 1 stop	7D = 5 data, 2 stop

<n>P — Set parity corresponding to n.

0P = none	2P = none
1P = odd	3P = even

(Note that the SCC 8530 does not support MARK and SPACE parity)

X<E/D> — Enable/Disable XON/XOFF handshaking protocol.
XE = Detect XOFF, await XON
XD = Ignore XOFF

F<E/D> — Enable/Disable keyboard input.
FE = Insert keystrokes into serial input stream
FD = Disable

L<E/D> — Add line feed after carriage return.
LE = Add linefeeds after each carriage return output
LD = Do not add line feeds after carriage return output

R — Reset the SCC and input/output hooks.

Z — Zap control character interpretation.

The following commands are for communications mode only:

E<E/D> — Echo input to screen.
EE = Echo input
ED = Don't echo in

M<E/D — Enable/Disable line feed filtering after carriage returns.

A<E/D> — Enable/Disable input buffering.

B — Transmit a 233 milllisecond break (all zeros).

T — Enter terminal mode.

Q — Exit terminal mode.

The following commands are for printer mode only:

<n>N — Page formatting (# of chars 1 to 255 before forced carriage return). (Note that a zero disables formatting.)

T<E/D> — Enable/Disable BASIC tabbing.
TE = Implement BASIC tabs
TD = Do not implement BASIC tabs

Handshaking

The Cortland serial ports use the Mini-Din 8 connector; this standard provides a simple handshake procedure. A character is transmitted when the transmit buffer is empty and HSKIN is asserted. To receive characters, HSKOUT is asserted (when a character is in the input buffer, it is received). The General-Purpose input line is readable with a Pascal interface status call.

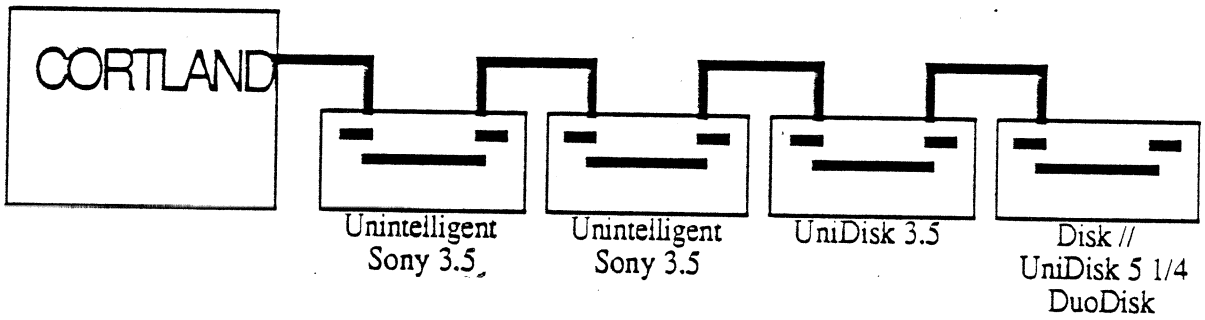
Chapter 3

Disk Support

Introduction

The Cortland disk-support system, with a built-in Integrated Woz Machine (IWM) chip, accommodates Disk II (Duodisks, Unidisks), Sony 3.5-inch drives with or without built-in intelligence (Unidisk 3.5). Port 6 is the standard Disk II support slot. Disk II boot routines are built into ROM. Disk II routines in DOS, ProDOS, and Pascal operate the same as they do in Apple II. Port 5 (internal slot 5) controls the intelligent and unintelligent Sony 3.5-inch drives as well as the RamDisk. You can attach up to two Disk IIs, two unintelligent Sony 3.5-inch drives, and two or more intelligent Sony 3.5-inch drives, depending on IWM output specifications. The disks must be attached as follows:

Note: Two unintelligent Sony 3.5-inch drives are shown below. This is the maximum number supported. There may be more than one Unidisk 3.5, or no Unidisk 3.5s attached where the Unidisk 3.5 is shown.



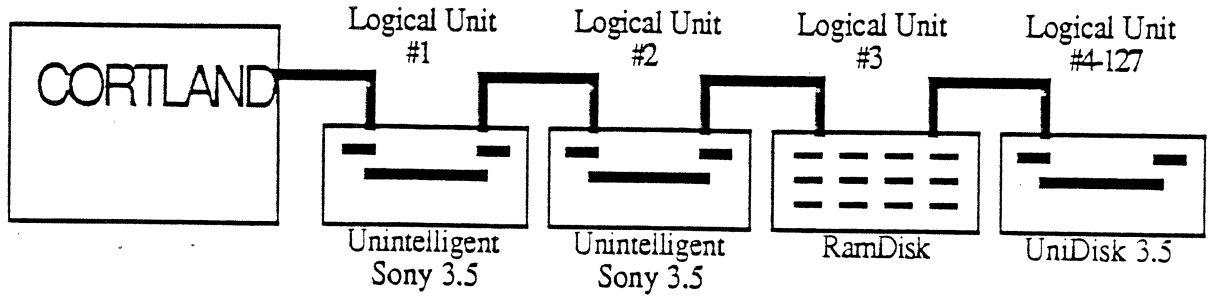
Port 5 and port 6 disk interface routines access the IWM, using slot 6 soft switches. The firmware arbitrates between slot use of the same soft switches. If a peripheral card is plugged into slot 6, the firmware in port 5 can still access the disks plugged into port 6's IWM connector by disabling the external peripheral card temporarily, performing the disk access, and then reenabling the external peripheral card.

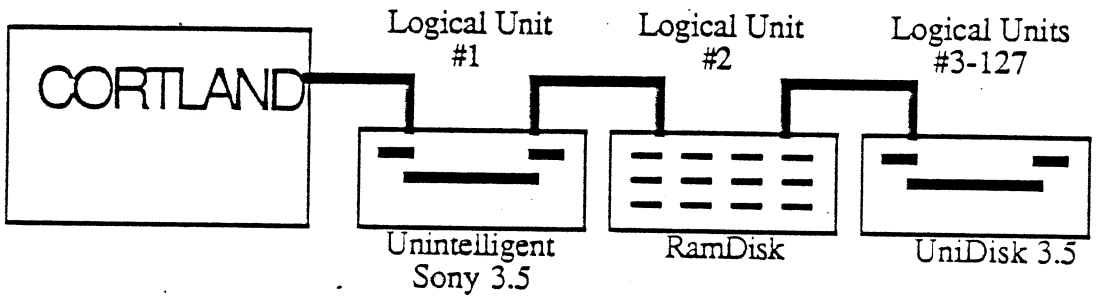
The port 5 disk interface for Unidisk 3.5 is called Smart Port. It consists of a superset of the Protocol Converter software used in the 32K Apple II ROM version. Smart port supports two unintelligent Sony 3.5-inch drives, the RamDisk, and the Unidisk 3.5 up to a total of 127 combined devices. The IWM can support up to six devices maximum.

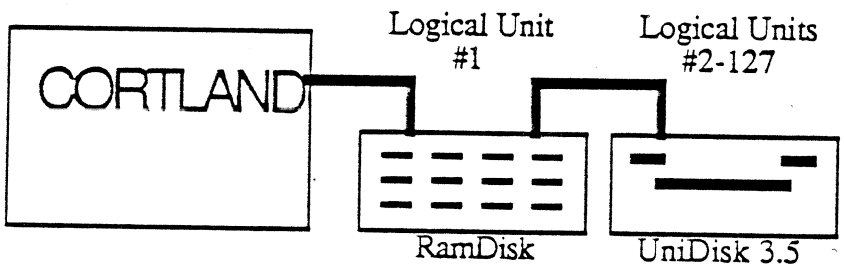
Slot 5 Initialization

During power up (initialization time), or during the slot-5 boot process, a reset of all devices supported by the slot 5 driver is initiated to be followed by a device ID assignment process. During the ID assignment process, the firmware determines the quantity of devices connected to the protocol converter bus, and assigns a logical unit number to each

device starting with a unit number of 1. Devices are assigned unit numbers starting with unintelligent Sony 3.5-inch drives, followed by the RamDisk, and then the Unidisk 3.5 devices. The logical location of devices on the protocol converter chain may differ from the physical location due to the assignment of the logical unit number just prior to the first Unidisk 3.5 device to the RamDisk as shown below.







During the device ID assignment process, the logical unit number assigned to the RamDisk, is saved in bank $E1. When the slot 5 driver is called, the driver compares the unit number, passed as one of the input parameters, to the RamDisk unit number. If the unit number is less than the RamDisk unit number, control is passed to the unintelligent Sony driver. If the unit number is equal to the RamDisk unit number, control is passed to the RamDisk driver. If the unit number is greater than the RamDisk unit number, control is passed to the Protocol Converter Bus driver.

Slot 5 Boot

A call to the slot 5 boot entry point forces all devices on the protocol converter chain to be reset, followed by the device ID assignment process. Then the boot block is read from the first logical device on the protocol converter chain. If the boot block is found, then a jump to $000801 occurs. If the boot block is not found and the boot was called from the boot scan routine (power up or auto boot), then control is passed back to the slot scan routine. If the boot was not from the scan routine, such as PR#5 or 00/C500G from the Monitor, then an error message is displayed to indicate the boot failure.

> Note: Some issues regarding the slot 5 boot process: If no unintelligent Sony drives are installed, it is not possible to boot from logical unit 1 or the RamDisk on power up. Although boots from the RamDisk could be avoided, it may be desirable to boot from the RamDisk after ProDOS has been installed via a PR#5.

All devices connected as a smart-port device, such as Unidisk 3.5, should respond to all protocol converter calls, and hand back appropriate status information.

Information specific to the RamDisk driver is covered in a separate document.

Dumb Sony

The Dumb Sony (Dsony) drive has an 800K-byte capacity, which is used in the Cortland system. This drive is a Unidisk 3.5 without the controller card. The Cortland contains all the intelligence required to access the drive. The Dsony behaves as a protocol converter device. The driver is accessed through the Cortland protocol converter firmware. Parameters are pased to the Dsony driver through nine bytes of zero page. The following depicts the parameter-passing area layout:

	X Reg = Unit Number
$42	Command
$43	param_count = 3
$44	Buffer_addr_low
$45	Buffer_addr_high
$46	Block_number_low
$47	Block_number_medium
$48	Block_number_high
$49	Spare Block number
$4A	Buffer_Bank_address

Determined by the specific call being made, some of the parameter field may not be required. On the exit from a call to the Dsony firmware, the accumulator contains a status byte indicating the success or failure of the call. If the call is successful, the Carry bit is cleared, otherwise; it ise set and the accumulator contains the error code.

The list of calls supported are:

- Status
- Read block
- Write block
- Format disk
- Control
- Init
- Read and write

The individual calls are described in the Protocol Converter specification.

Chapter 4

Cortland Mouse

Introduction

The Cortland Mouse, also known as the FDB Mouse, uses the Front Desk Bus (FDB) to communicate with the Cortland Keyboard micro; the micro informs the monitor firmware of Mouse actions. For those who are familiar with Apple II, this is a departure from the Apple II Mouse interface which depends on firmware to support the Mouse. Cortland's Mouse has a true passive mode like the Apple II; however, the Apple II Mouse requires interrupts to function. The true passive mode is an advantage, in that, it allows Mouse applications to operate and allows devices to operate that have timing-critical loops, or that can't run if interrupted. The true passive mode also prevents the 65816 from being overburdened with interrupts from the Mouse.

Interrupts

The FDB Mouse can cause an interrupt to the 65816 only if an interrupt mode has been selected: the mouse on and the interrupt condition has occurred. For those familiar with Apple II, it interrupts when the Mouse is moved. The FDB Mouse interrupts in synchronization with Cortland's Vertical Blanking signal (VBL). The Mouse can interrupt the 65816 a maximum of 60 times per second. This cuts down on the burden the Mouse puts on the 65816. If an interrupt condition (determined by the mode-byte setting) occurs, an interrupt will be passed to the Key Glu chip to cause it to interrupt the 65816 in synchronization with the VBL if a predefined condition exists.

Features

The Mouse provides position data that returns a position change of up to +/- 63 counts or approximately 0.8 inches of travel. Cortland's firmware converts this relative-position data to an absolute position. The FDB Mouse also provides the following features:

- Current button 0 and button 1 data (1 if down; 0 if up)
- Previous button 0 and button 1 data (1 if it was down; 0 if it was up)
- Interrupt data (whether VBL, button 0/1, or movement interrupt)

At power up, the FDB Mouse, by way of the Key Glu chip, defaults to a Mouse-off non-interrupt state. Reset will cause the Key Glu chip to turn the FDB Mouse interrupt off and enter a non-interrupt state.

Addresses Used

Address	Function
$C027	Key Glu status register defined as follows: Bit 0 = d Must not be altered by Mouse Bit 1 = 0 'X' position available (read only) = 1 'Y' position available (read only) Bit 2 = k Must not be altered by Mouse Bit 3 = k Must not be altered by Mouse Bit 4 = d Must not be altered by Mouse Bit 5 = d Must not be altered by Mouse Bit 6 = 1 Mouse interrupt enable (read/write) Bit 7 = 1 Mouse register full (read only) k = used by keyboard handlers d = used by FDB handlers
$C024	Mouse data register: 1st read yields 'X' position data and button-1 data 2nd read yields 'Y' position data and button-0 data

Program Requirements and Restrictions

- To enable Mouse interrupts, set bit 6 of location $C027 to 1.
- To determine if an interrupt came from the Mouse, read bit 7 of $C027 and bit 6 of $C027. If both = 1, then an interrupt is pending from the Mouse.
- To read the Mouse position, the following conditions must occur or the data are contaminated and corrective measures must be taken:

1. Read bit 7 of $C027:
   If bit 7 = 0, then 'X' and 'Y' data are not yet available
   If bit 7 = 1, then data are available but can be contaminated

2. Read bit 1 of $C027 only if bit 7 = 1:

   If bit 1 = 0, then 'X' and 'Y' data are not contaminated and can be read. The first read of $C024 returns 'X' data and button-1 data; the second read of $C024 returns 'Y' data and button 0 data.

   Caution must be observed when using indexed instructions. The false read and write results of some indexed instructions can cause the 'X' data to be lost and the 'Y' data to appear when 'X' data was expected.

   If bit 1 = 1 and $C024 has not been read, then the data in $C024 are contaminated and must be considered useless. If that condition occurs, perform the following steps:

   - Read $C024 one time only.

- Ignore the byte read in.

- Exit your Mouse-read routine without updating the 'X' and 'Y' or button data. This will not harm any program; however, it guarantees that the next time you read Mouse positions they will be accurate.

3. The data read in are encoded as follows:

'X' data byte:
If bit 7 = 0, then Mouse button 1 is up.
If bit 7 = 1, then Mouse button 1 is down.

Bit 0-6 delta Mouse move:

If bit 6=0, then a positive move up to $3F (63).
If bit 6=1, then a negative move in two's complement up to $40 (64).

'Y' data byte:
If bit 7 = 0, then Mouse button 0 is up.
If bit 7 = 1, then Mouse button 0 is down.

Bit 0-6 delta Mouse move:

If bit 6 = 0, then a positive move up to $3F (63) ticks.
If bit 6 = 1, then a negative move in two's complement up to $40 (64).

- The main screen holes can be in either bank $00 or bank $E0, determined by whether shadowing is on or off. If shadowing is on, the screen holes are in bank $00; if shadowing is off, the screen holes are in bank $E0.

Main Memory Screen

The Mouse is resident in Cortland's internal slot 4. When the Mouse is in use, the main memory screen holes for slot 4 hold 'X' and 'Y' absolute position data, current mode, button 0/1 status, and interrupt status. Eight additional bytes are used for Mouse information storage. They hold the maximum and minimum clamps for the Mouse's absolute position. The following lists the Mouse's screen-hole use when Cortland firmware is used:

Address	Use
$47C	Low byte of absolute X position
$4FC	Low byte of absolute Y position
$57C	High byte of absolute X position
$5FC	High byte of absolute Y position
$67C	Reserved and used by firmware
$6FC	Reserved and used by firmware
$77C	Button 0/1 interrupt status Bit 7 = Currently button 0 is up/down (0/1) Bit 6 = Previously button 0 was up/down (0/1)

Bit 5 = X/Y moved since last READMOUSE
Bit 4 = Currently button 1 is up/down (0/1)
Bit 3 = VBL interrupt
Bit 2 = Button 0/1 interrupt
Bit 1 = Movement interrupt
Bit 0 = Previously button 1 was up/down (0/1)

Interrupt Status Byte

Current Button 0 status	Previous Button 0 status	X/Y move since 1st Read	Current Button 1 status	VBL interrupt occurred	Button interrupt occurred	Movement interrupt occurred	Previous Button 1 status

$7FC Mode byte
Bit 7 = Reserved
Bit 6 = Reserved
Bit 5 = Reserved
Bit 4 = Reserved
Bit 3 = Interrupt on VBL
Bit 2 = Interrupt on next VBL if button pressed
Bit 1 = Interrupt on next VBL if Mouse moved
Bit 0 = Mouse off/on (0/1)

Mode Byte

Reserved	Reserved	Reserved	Reserved	VBL interrupt mode	Button interrupt mode	Movement interrupt mode	Mouse off/on

Firmware RAM

The Mouse clamps reside in the following auxiliary screen-hole locations:

Address	Use
$E0xxxx	Low byte of low 'X' clamp
$E0xxxx	High byte of low 'X' clamp
$E0xxxx	Low byte of high 'X' clamp
$E0xxxx	High byte of high 'X' clamp
$E0xxxx	Low byte of low 'Y' clamp
$E0xxxx	High byte of low 'Y' clamp
$E0xxxx	Low byte of high 'Y' clamp
$E0xxxx	High byte of high 'Y' clamp

You must never attempt to change these locations directly; they must be changed using CLAMPMOUSE.

Firmware Calls

To use the Mouse firmware, enter by way of the user interface provided below. This interface conforms to the Pascal 1.1 protocol for peripheral cards.

Location	Routine	Definition
$C40D	PINIT	Pascal INIT device (Not implemented)
$C40E	PREA	Pascal READ character (Not implemented)
$C40F	PWRITE	Pascal WRITE character (Not implemented)
$C410	PSTATUS	Pascal get-device status (Not implemented)
$C411 = $00		Indicates that more routines follow

Standard routines that are implemented on Cortland, Apple II, and AppleMouse card.

$C412	SETMOUSE	Set Mouse mode.
$C413	SERVEMOUSE	Service Mouse interrupt.
$C414	READMOUSE	Read Mouse position.
$C415	CLEARMOUSE	Clear Mouse position to 0 (for delta mode).
$C416	POSMOUSE	Set Mouse position to user-defined position.
$C417	CLAMPMOUSE	Set Mouse bounds in a window.
$C418	HOMEMOUSE	Set Mouse to upper-left corner of clamping window.
$C419	INITMOUSE	Reset Mouse clamps to defaults, positions to 0,0.
$C400	BINITENTRY	Initial entry point when coming from BASIC
$C405	BASICINPUT	BASIC input entry point (opcode SEC) Pascal ID byte
$C407	BASICOUTPUT	BASIC output entry point (opcode CLC) Pascal ID byte
$C408 = $01		Pascal generic signature byte
$C40C = $20		Apple Tech Support ID byte
$C4FB = $D6		Additional ID byte

Pascal Firmware Calls

Pascal recognizes the Mouse as a valid device, however, Pascal is not directly supported by the firmware. A Pascal driver for the Mouse is available from Apple to interface programs with the Mouse. The standard Pascal calls PINIT, PREAD, PWRITE, and PSTATUS returns with the 'X' register set to 3 (Pascal illegal operation error) and carry set. The following is a list of the Pascal firmware calls:

PINIT

Function:	Not implemented (just an entry point in case user calls it by mistake).
Input:	All registers and status bits
Output:	X = $03 -- Error 3 = Bad mode: illegal operation C = 1 -- Always Screen holes: Unchanged

PREAD

Function:	Not implemented (just an entry point in case user calls it by mistake).
Input:	All registers and status bits
Output:	X = $03 -- Error 3 = Bad mode: illegal operation C = 1 -- Always Screen holes: Unchanged

PWRITE

Function:	Not implemented (just an entry point in case it's called by mistake).
Input:	All registers and status bits
Output:	X = $03 -- Error 3 = Bad mode: illegal operation C = 1 -- Always Screen holes: Unchanged

PSTATUS

Function:	Not implemented (just an entry point in case user calls it by mistake).
Input:	All registers and status bits
Output:	X = $03 -- Error 3 = Bad mode: illegal operation C = 1 -- Always Screen holes: Unchanged

Assembly-Language Firmware Calls

To use a Mouse routine from assembly language, read the location corresponding to the routine you want to call. The value read is the offset of the entry point of the routine to be called.

The following lists the available Cortland firmware calls:

Notes:

1. n = Mouse slot number

2. The following bits are not changed by Mouse firmware:
   - e, m, I, x
   - Direct register
   - Data bank register
   - Program bank register

3. Mouse screen holes are not to be changed except during POSMOUSE when new Mouse coordinates are put directly into the screen holes. No other Mouse screen hole can be changed without adversely affecting the Mouse.

4. If shadowing is on, use the screen holes in bank.$00. If shadowing is off, use the screen holes in bank $E0.

SETMOUSE

Function: Sets Mouse operation mode.
Input: A = mode ($00 to $0F, only valid modes)
X = Cn for standard interface (Cortland Mouse not affected)
Y = n0 for standard interface (Cortland Mouse not affected)
Output: A = mode if illegal mode entered, else A is scrambled
X, Y, V, N, Z = scrambled
C = 0 if legal mode entered (mode is <= $0F)
C = 1 if illegal mode entered (mode is > $0F)
Screen holes: Mode byte updated only.

SERVEMOUSE

Function: Tests for interrupt from Mouse, and resets Mouse's interrupt line.
Input: A, X, Y = not affected
Output: X, Y, V, N, Z = scrambled
C = 0 if it was a Mouse interrupt
C = 1 if it was not a Mouse interrupt
Screen holes: Interrupt status bits updated to show current status.

READMOUSE

Function: Reads delta (X/Y) positions, updates absolute X/Y positions, and reads button statuses from FDB Mouse.
Input: A = not affected
X = Cn for standard interface (Cortland Mouse not affected)
Y = n0 for standard interface (Cortland Mouse not affected)
Output: A, X, Y, V, N, Z = scrambled
C = 0--Always
Screen holes: SLO, XHI, YLO, YHI buttons and movement status bits updated--interrupt status bits are cleared.

CLEARMOUSE

Function: Resets to 0, X, and Y, the buttons, movement, and interrupt status. This mode is intended for delta Mouse positioning instead of the normal absolute positioning.
Input: A = not affected
X = Cn for standard interface (Cortland Mouse not affected)
Y = n0 for standard interface (Cortland Mouse not affected)
Output: A, X, Y, V, N, Z = scrambled
C = 0--Always
Screen holes: SLO, XHI, YLO, YHI buttons and movement status bits updated--interrupt status bits are cleared.

POSMOUSE

Function:	Allows user to change current Mouse position.
Input:	User places new absolute X/Y positions directly in appropriate screen holes. X = Cn for standard interface (Cortland Mouse not affected) Y = n0 for standard interface (Cortland Mouse not affected)
Output:	A, X, Y, V, N, Z = scrambled C = 0--Always Screen holes: User changed X and Y absolute positions only--bytes changed.

CLAMPMOUSE

Function:	Set up clamping window for Mouse use. Power up defaults are 0 to 1023 ($0000-$03FF).
Input:	A = 0 if entering X clamps A = 1 if entering Y clamps Clamps are entered in slot 0 screen holes by the user as follows: • $478 = low byte of low clamp • $4F8 = low byte of high clamp • $578 = high byte of low clamp • $5F8 = high byte of high clamp X = Cn for standard interface (Cortland Mouse not affected) Y = n0 for standard interface (Cortland Mouse not affected)
Output:	A, X, Y, V, N, Z = scrambled C = 0 -- Always Screen holes: X/Y absolute position set to upper-left corner of clamping window. Clamping RAM values in bank $E0 are updated.

The Cortland Mouse performs an automatic HOMEMOUSE after a CLAMPMOUSE. This is not through either the AppleMouse card or the Apple II. After executing a CLAMPMOUSE, follow immediately with the execution of a HOMEMOUSE when dealing with the Apple II or the AppleMouse. The execution of a HOMEMOUSE is required because the delta information is being fed to the firmware instead of +/-1's as is the case for the Apple II and the 6805 AppleMouse microprocessor card. The delta information from Cortland's FDB Mouse can alter the absolute position to a point where the clamping techniques used by the other two mouse devices are useless for Cortland.

HOMEMOUSE

Function:	Sets X/Y absolute positiion to upper-left corner of clamping wndow.
Input:	A = not affected X = Cn for standard interface (Cortland Mouse not affected) Y = n0 for standard interface (Cortland Mouse not affected)
Output:	A, X, Y, V, N, Z = scrambled C = 0--Always Screen holes: User changed X and Y absolute positions only--bytes changed.

INITMOUSE

Function:	Sets screen holes to defaults, and sets clamping window to default of 0000-1023 ($0000, $03FF) in both the X and Y directions. Resets Key Glu Mouse interrupt capabilities.
Input:	A = not affected X = Cn for standard interface (Cortland Mouse not affected) Y = n0 for Standard interface (Cortland Mouse not affected)
Output:	A, X, Y, V, N, Z = scrambled C = 0--Always Screen holes: X/Y positions, button statuses, interrupt status reset.

Note: Button and movement statuses are valid only after a READMOUSE. Interrupt status bits are valid only after a SERVEMOUSE. Interrupt status bits are reset after a READMOUSE. Read and use, or read and save the appropriate Mouse screen-hole data before enabling or reenabling 65816 interrupts.

Standard Firmware Call Example

Note: Interrupts must be disabled on every call to the Mouse firmware.

```
SETMOUSEOFF  EQU  $Cn12          ;Offset to SETMOUSE offset ($C412 for
                                  Cortland).
             LDA  SETMOUSEOFF    ;Get offset into code
             STA  TOMOUSE+1      ;Modify operand
             LDX  Cn             ;Where Cn = C4 in Cortland
             LDY  n0             ;Where n- = 40 in Cortland
             PHP                 ;Save interrupt status
             SEI                 ;Guarantee no interrupts during call
             LDA  #$01           ;Turn Mouse passive mode on
             JSR  TOMOUSE        ;JSR to a modified JMP intruction
             BCS  ERROR          ;C = 1 if illegal-mode-entered error
             PLP                 ;Restore interrupt status
             RTS                 ;Exit
ERROR        PLP                 ;Restore interrupt status
             JMP  ERRORMESSGE    ;Exit to error routine
TOMOUSE      JMP  $Cn00          ;Modified operand for correct entry point
                                  $C400 for Cortland
```

BASIC Firmware Entry

The Mouse and BASIC have the following interface. To turn the Mouse on, perform the following steps:

```
1. PRINT CHR$(4);"PR#4"  :REM Ready Mouse for output
2. PRINT CHR (1)         :REM Send the Mouse a 1 to turn it on from BASIC
3. PRINT CHR$(4);"PR#0"  :REM Restore screen output.
```

Note: Use PRINT CHR$(4);"PR#3" to return to 80 columns.

To accept outputs from BASIC, the firmware changes the output hooks at $36 and $37 to point to $C407 and performs an INITMOUSE (described above).

To turn the Mouse off, perform the following steps:

```
1. PRINT CHR$(4);"PR#4"  :REM Ready Mouse for output
2. PRINT CHR (0)         :REM Send the Mouse a 1 to turn it off from BASIC
3. PRINT CHR$(4);"PR#0"  :REM Restore screen output.
```

Note: Use PRINT CHR$(4);"PR#3" to return to 80 columns.

To read Mouse position and button statuses from BASIC, perform the following steps:

```
1. PRINT CHR$(4) "IN#4"  :REM  Ready Mouse for input
2. INPUT X, Y, B         :REM  Input Mouse position
3. PRINT CHR$(4) "IN#0"  :REM  Return keyboard as input device when reading
                                Mouse positions has been completed.
```

When the Mouse is turned on from BASIC (to input data), the firmware changes the input hooks at $38 and $39 to point to $C405. When an INPUT statement is invoked while talking to the Mouse, the firmware performs a READMOUSE before converting the screen hole data to decimal ASCII and placing it in the input buffer at $200.

In BASIC, the Mouse runs in passive mode or a non-interrupt mode. Clamps are set automatically to 0000-1023 ($0000-$03FF) in both the X and Y directions, and position data in the screen holes are set to 0.

During a BASIC INPUT statement, the firmware reads the position changes (deltas) from the FTD Mouse, adds them to the absolute position in the screen holes, clamping the positions if necessary, and converts the absolute positions in the screen holes to ASCII. The firmware then places that data, with the button 0 status, into the input buffer followed by a carriage return and returns to BASIC.

Button 1 status cannot be returned to BASIC since that would add another input variable to the input buffer resulting in an ?EXTRA IGNORED error being printed in the existing Mouse BASIC program. A BASIC program, wanting to read button 1 status, can PEEK the screen hole containing that data. The data returned in the input buffer is in the following form:

```
s x1 x2 x3 x4 x5 , s y1 y2 y3 y4 y5 , sb B0 b5 cr
```

```
        s = sign of absolute position
x1....x5 = 5 ASCII characters giving the decimal value of X
y1....y5 = 5 ASCII characters giving the decimal value of Y
      sb = - if key on keyboard was pressed during input statement
           + if no key was pressed during input statement
      B0 = ASCII space character
      b5 = 1 if button 0 is pressed now and was pressed on last INPUT statement
         = 2 if button 0 is pressed now but was not pressed on last INPUT statement
         = 3 if button 0 is not pressed now but was pressed on last INPUT statement
         = 4 if button 0 is not pressed now and was not pressed on last INPUT statement
      cr = Carriage Return--required as terminator before passing control from firmware
           back to BASIC.
```

Note: The BASIC program must reset the key strobe at $C010 if sb returns to a negative state. A POKE 49168,0 resets the strobe.

Chapter 5

Single-Chip Microcomputer Keyboard Interface (SKI)

Introduction

The Front Desk Bus (FDB) and the internal keyboard are controlled by software protocols between the system processor and the Single-Chip Microcomputer. This chapter describes these software protocols.

SKI Devices

The following devices comprise the SKI:

Microcomputer (uC) Chip

The uC is a single-chip micro with three basic functions:

- Scans the built-in (internal) keyboard and periodically polls FDB for keyboard and keypad data.
- Acts as the FDB host for the mouse by periodically polling the FDB mouse.
- Acts as a transceiver chip for other FDB devices. The system tells the chip to issue listen/talk commands on FDB.

The uC can be interrupted or polled by the system, but it may not respond for up to 4.5 ms if it has started an FDB operation. FDB operations cannot be interrupted once they have begun or data will be lost.

KEY GLU

Key Glu allows communication between the uC and the system processor. The chip acts as a holding register so that data written by the uC can be read by the system and data written by the system can be read by the uC. This chip is also used to generate interrupts to the system, and to aid in performing the internal keyboard scan.

The Keyboard

The uC processes all keyboard operations by scanning the built-in keyboard and FDB for keypresses. All keystrokes are passed back to the system using the same method as in the Apple II. If an FDB keyboard or keypad is connected to the system, the uC acts as the

FDB host and automatically reads keystrokes from the devices. The keyboard matrix is the same as the one implemented on the Apple II (80 positions) so that the retrofit board can use the existing Apple II keyboard and keypad.

The SKI performs the following steps during normal keyboard operation:

Scanning for Keystrokes

Scanning the built-in keyboard consists of checking for keypresses and converting them into the proper ASCII code. Auto-repeat rate is selectable: no repeat or 40, 30, 24, 20, 15, 11, 8, or 4 keystrokes per second.

The keyboard will only auto-repeat as fast as keys are being read. If the buffer (normally 1 key, unless the buffer mode selected is that which employs a 16-key buffer) is not empty, then an auto-repeat key will not be put in the buffer (this prevents the cursor, etc. from jumping immediately after long operations, such as disk accesses). The delay before auto-repeat is also selectable: 1/4, 1/2, 3/4, and 1 second. The keyboard scan attempts to implement the same idiosyncrasies as the current keyboard encoder (1-key buffer, pseudo N-key rollover, including ghosting and phantom keys).

International keyboard layouts are identified at power up by reading a specific location in the battery-backed RAM. A command can be executed to change the current layout. On power up, the uC uses the keyboard layout specified by a command from the system. On reset, the uC uses the last layout specified by the software or system menu. The FDB keyboards have a key labeled with a period (.), which is not used on international keyboards because some languages use the comma (,) instead. Each keyboard layout is preset to default to either the period (.) or the comma(,) as follows:

- The U. S., U. K., Dvorak, and Canada use the period (.).
- France, Denmark, Spain, Italy, Germany, and Sweden use the comma (,).

It is possible to override the default by setting a specific bit when indicating the keyboard layout and character set to be used. This bit will swap the setting to the opposite of the preset default.

A new mode, called the Dual Speed mode, doubles the auto-repeat rate for the four arrow keys when CONTROL is pressed; this mode is always enabled. An optional extension of this mode allows you to double the repeat rate of the delete key and the space bar when CONTROL is pressed. This mode extension is enabled using the setup menu/control panel. Another optional mode allows you to repeat at four times the normal repeat rate.

Poll FDB

All FDB keyboards and keypads are automatically processed by the uC. Keystrokes read from FDB keyboards/keypads are incorporated into the normal stream of keystrokes detected on the built-in keyboard. A command that disables the automatic FDB keyboard/keypad poll is implemented so that a multi-player game that requires many keypads can be used.

Keystrokes

Keypresses are returned by loading keylatch with data. Normal keyboard operations are compatible with the Apple II keyboard. The keylatch is read at address $C000, with the

msb indicating whether the key is valid (KYSTB). AKD is read on msb of address $C010 and the KYSTB is cleared by reading $C010 or writing $C01X.

Key Modifiers

Key modifiers, such as SHIFT, CONTROL, CAPS LOCK, OPEN-APPLE, SOLID-APPLE, Auto-repeat, and Keypad are stored in the key-modifier register. The key-modifier register is updated when a key is pressed and the ASCII value is loaded into the keylatch. The values stored in the key-modifier register reflect the state of the key modifiers when the key was pressed and not the current state of the modifiers. This allows a program to read the keylatch and modifiers after a disk operation and detect if OPEN-APPLE was pressed when the key was pressed. Currently, if a key and OPEN-APPLE are pressed during a disk operation, a program will detect that a key was pressed, but may miss the OPEN-APPLE since it was let up before the disk access was finished. (Keystrokes are not read during ProDOS operations.)

Table 1. Key Modifiers

Bit	Modifier
0	SHIFT
1	CONTROL
2	CAPS LOCK
3	Repeat
4	Keypad
5	Updated Modifier latch without keypress
6	SOLID-APPLE
7	OPEN-APPLE

Bit 5 (update bit) signifies that the modifier register was changed without any other keypresses occuring. This only occurs when the KYSTB is clear. For example, if only CONTROL or SHIFT is pressed and the KYSTB is clear, then the uC will indicate this by setting the update bit and changing the status of the control or shift bit in the modifier register. When a new key is pressed, such as 'x', then the modifier register is updated along with the keylatch (and KYSTB is set) and the update bit is cleared. The modifier register will be updated in two cases: when a new key is written into the keylatch (with the update bit cleared), and if the KYSTB is clear and a modifier condition changes (with the update bit cleared).

Appendix C lists the keycodes generated by the FDB keyboard. Codes 96 through 126 are extra, undefined codes for the FDB keyboards. These codes are processed by passing them directly through the keyboard latch with the keypad bit set. These codes can then be used as macro keys, software-defined keys, or function keys. Code 127 ($7F) is reserved for reset (not to be confused with keypad key 64, DELETE , which is translated into ASCII code $7F with the keypad bit set).

The Open- and SOLID-APPLE bits are needed so that the uC knows when to drive the Open or Solid Apple outputs. These outputs are driven high before the key is sent to simulate someone pressing the Apple keys. This allows the system to emulate the existing keyboard with the FDB keyboard.

Since the current keyboard is unbuffered and allows an overrun to occur, the key-modifier register is not always valid (unless the keyboard is in buffering mode). There is a small window of time when the keylatch has key1, while the key-modifier register has modifiers to key2. (The key modifiers must be written out first, since keylatch sets the KYSTB, indicating both bytes are valid.) To verify that the key-modifier register is accurate, both the keylatch and key-modifier registers must be read until the data in each register is the same for two successive times. The second pair of register reads should be at least 30 usec apart. (If the keyboard has been placed in the buffer mode, then the modifier byte is valid after the KYSTB is set.) The different methods of reading the keyboard are described later in this document.

Keyboard Interrupt Mode

The key Glu chip can be set up to interrupt whenever the keylatch is written into by the uC. This interrupt can be cleared with two different operations. Typically, software reads a key, then clears the interrupt by clearing the KYSTB. But occasionally certain tasks, such as background routines wants to intercept a key before it gets to the keyboard. If the software installs an interrupt handler that looks for a specific keypress, the handler must always clear the interrupt, regardless of whether it reads the keyboard, or the system will hang in an infinite loop. If the handler chooses to ignore the current key and not clear the KYSTB, then it can clear the interrupt by reading the keyboard.

Keyboard Buffering Mode

A buffer mode exists where the uC buffers keystrokes and modifiers. The uC sends a new keystroke and modifiers when the KYSTB is cleared (also, the Flush buffer command). In this mode, the system polls the keylatch until the msb is set, then reads the key modifier register, and finally clears the KYSTB. Both registers will stay valid until the KYSTB is cleared.

Reset and the Keyboard

The uC periodically samples the RESETin line to determine if it should reset the system (using the RESETout signal). If it detects that RESETin is low, then the uC will check if both CONTROL and RESET have been pressed on either the internal or the FDB keyboard before setting RESETout low. If CONTROL has not been pressed, then the uC will continue sampling the internal keyboard and FDB, but will only set RESETout if RESETin is still low while CONTROL is pressed.

When RESETin is released, RESETout will also be released. If the FDB OPEN-APPLE key had been pressed, then the uC will simulate this by pulling the OAPLout line high. The system firmware will detect this and institute a power-up reset sequence, that includes a software reset command to the uC, which will then reset itself and FDB. Even though RESET is pressed, the keyboard must continue to scan both the internal keyboard and the FDB keyboard so that the status of the Apple keys can be maintained.

The keyboard is read by the system using one of the following modes:

Apple II Mode

This mode is compatible with the existing Apple II, and it is unbuffered and asynchronous. The following are the function locations:

- Reads keyboard data at $C000
- Clears keyboard strobe at $C010
- OPEN- and SOLID-APPLE are read at soft switch locations $C061 and $C062.

Apple II Mode with Key Modifiers

This mode emulates existing Apple II, but extends the keyboard data by including keyboard modifiers. The bits in the modifier register which represent the status of the Apple keys may not reflect the same state as the Apple key locations $C061 and $C062. When the uC is running with the keyboard unbuffered, the Apple key soft switch values always reflect the state of the Apple key inputs. The Apple bits in the modifier register may not be the same because the modifier register is updated as follows:

- When the keylatch is updated with a new ASCII code.

- If no keys are down and the KYSTB is clear. The modifier register is then updated approximately every eight ms.

In the unbuffered mode, the uC updates the keylatch and modifier register asynchronously to the system. To determine if the data in the key modifier register is accurate, use the following procedure:

1. If bit 5 (updated without keypress) is set, then the register contents are accurate.

2. If bit 5 is not set, then both the keyboard latch and the modifier register must be read (30us apart) until the data in each is the same for two successive times.

Buffered Apple II Mode

This mode emulates the Apple II mode with key modifiers, but only sends new keystrokes and modifiers after the KYSTB has been cleared. To use this mode properly, both bytes, the keyboard latch and the modifier register must be read, while the Apple key locations ($C061, $C062) must be ignored. The program looks for keystrokes by waiting until the KYSTB bit (msb of keyboard latch) is set before reading the keyboard and modifier register. After reading both bytes, the keyboard strobe is cleared to indicate that the program is ready for the next keystroke.

In this mode, an application program detects modifier keys, such as CONTROL, SHIFT, or LOCK. Normally, the modifier register reflects the state of the modifier keys when another key is pressed. However, if no keys are down (which can be detected by checking the AKD flag on the msb of location $C010), the KYSTB is cleared, and the update bit 5 of the modifier register is set; the modifier byte has then been updated to reflect the current state of the modifiers. If another keypress occurs, then the update will be cleared, both the keylatch and modifier registers will be updated, and the KYSTB is set. The KYSTB must be cleared before the modifier register is updated.

While you can switch in this mode with existing software to prevent the system from missing keystrokes when an overrun occurs, it has certain side effects. Som existing

software will automatically clear the strobe at certain times, such as coming back from a disk access. In this mode, the automatic clear will only clear the first key of a string of keystrokes. Also, since the buffer has no control over OPEN-APPLE and SOLID-APPLE, existing software, such as games, that reads the hardware locations ($C061, $C062) may not interpret the Apple keys properly. In this buffer mode, Apple-key soft switch locations $C061 and $C-62 reflect the state of the Apple key bits in the modifier register.

The buffer can be flushed by pressing the following key sequences:

OPEN-APPLE-CONTROL-DELETE.

The Front Desk Bus Mouse

The FDB mouse is processed automatically by the uC. The uC will periodically poll the FDB mouse to check for mouse activity. If the mouse has moved, or the button pushed, it will respond to the FDB mouse poll by returning two bytes of data. The uC will return this data to the system by writing both mouse data bytes to the Key Glu chip (Mouse byte Y followed by byte X--this enables the interrupt). The system then checks the status register to verify that a mouse interrupt have taken place, the two data bytes had been read, and mouse latch Y was read first. Key Glu clears the interrupt when the second latch has been read. To prevent overruns, the uC only writes mouse data when the registers are empty (i.e., after mouse latch X has been read by the system).

The advantages of this protocol is that the system is only interrupted at VBL time if the mouse has moved. This keeps the number of interrupts, and therefore the system overhead, to a bare minimum.

The uC won't perform another FDB mouse poll until both bytes have been transferred to the system.

Part of the initialization protocol sends the FDB address of the device to be automatically polled. While this address will typicallty indicate the FDB mouse as the polled device, it is possible to specify some other FDB device (with one caveat: whichever device is picked must transfer only 2 bytes. The uC will ignore all data sent by the mouse device if more than 2 bytes are sent, since there is no way to handle more automatically).

Table 2 depicts the 16 bits returned by the FDB mouse:

Table 2. FDB Mouse Data Bits

Bit	Function
15	Button pressed
14-8	Y-Delta movement (negative=up, positive=down)
7	'1'
6-0	X-Delta movement (negative=left, positive=right)

Additional Front Desk Bus Commands

All other FDB devices will be polled when the system sends an FDB poll token to the uC. In this mode, the uC acts as a dumb transceiver for FDB activity. This command protocol requires that the system specify the FDB command byte to be transmitted. The uC will transmit the byte, then wait for the FDB response. The uC returns data by storing a token in the data latch identifying the data that will follow; then it sends a new data byte each time the system reads the previous one.

If the token stored by the system indicates an FDB listen command, the uC reads all data bytes before initiating the FDB operation. The uC suspends all other operations until it recevies all data bytes.

The FDB response token stored in the data latch indicates to the system that the uC is responding to an FDB command. This token contains status bits that indicate if the FDB device responded (data valid), how many butes are coming (typically 2), and if a Service Request (SRQ) on FDB was detected. For example, only one byte is sent back if there was no response to the FDB poll. This byte is the response token which indicates no response and SRQ status.

When the token byte of a multi-byte reponse is stored in the data latch, the uC waits for 1 millisecond for the system to read the first data byte. This allows the system to read the FDB data back quickly if the data-latch interrupt and system interrupts are enabled.

If an FDB Service Request is detected and none of the auto-poll devices (keyboard, keypad, or mouse) are causing the interrupt, then the SRQ token is written into the data latch register. The SRQ token indicates, by setting the SRQ status bit, that some FDB device is currently requesting service. The system must start polling FDB devices when this bit is set, by sending FDB poll commands to the uC. A device requesting service will respond with data when it is polled.

If the system receives an FDB response with the SRQ status clear, then it should assume that there are no FDB devices requiring service. The opposite is not true and it may be possible for an SRQ set status, which disappears later, to be passed to the system. The system must be prepared to receive and handle spurious SRQ's. For example, if data is returned from a poll of a device which is not auto-polled and SRQ is also set, then the SRQ may be from an auto-poll device and could disappear when the uC automatically performs a poll of that device.

The system can send data to FDB devices (FDB listen mode) by sending the FDB LISTEN token to the uC, Followed by the command byte, then two data bytes. This transaction uses the DMD/DATA latch to transfer the bytes from the system to the uC.

Additional uC Commands

Abort Command. If the system passes the abort command to the uC, the uC flushes out any active commands. All commands that require the uC to transfer data to the system using the data latch are abruptly terminated.

Data Returned by the uC

One bit is reserved to signify that a special key sequence has been presssed. This bit can be used by the desktop manager, desktop accessories, or a switcher program. Using the data latch, rather than using the keyboard latch, allows the uC to interrupt the system and indicate this condition without also sending a dummy keystroke.

Boot Sequence Protocol

At boot time, the uC performs some preliminary initialization and then attempts to synchronize itself with the system by waiting for the SYNCH command from the system. All other commands are ignored. If the uC doesn't receive the SYNCH command within 1.5 seconds, it uses its built-in defaults. The defaults are:

Mode byte	All modes clear (see Appendix B, Command 5)
Delay before auto-repeat	3/4 sec.
Repeat rate	15/sec.
Language and layout	U. S.
FDB Keyboard address	$02
FDB Mouse address	$03

After this initialization, the system can change the defaults by using the change-configuration-bytes command.

Chapter 6

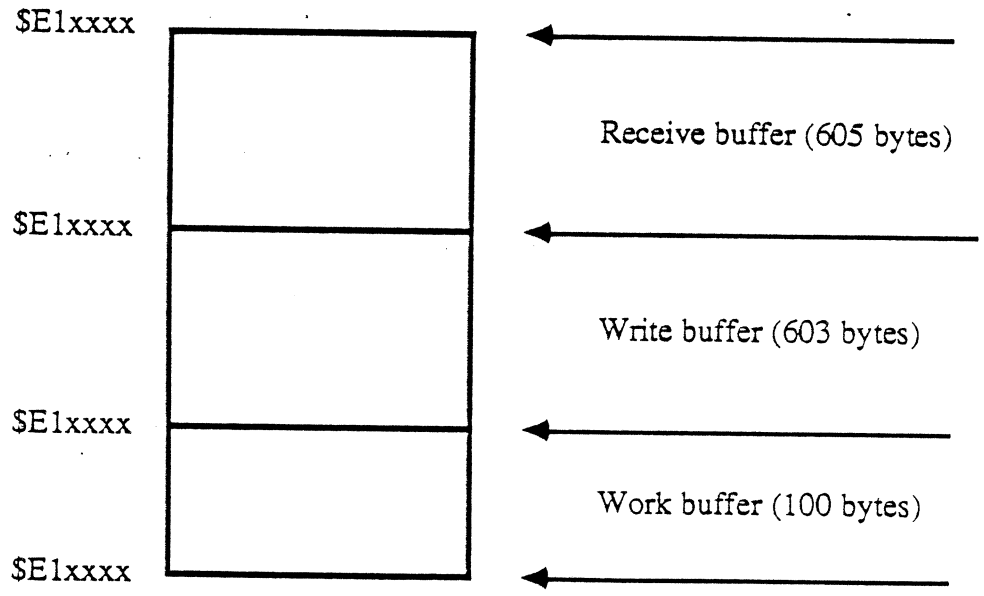
AppleTalk

Introduction

AppleTalk is a stand-alone work-area network that provides communications and resource sharing with up to 32 computers, disks, printers, modems, and other peripherals. AppleTalk consists of communications hardware and a set of communications protocols. This hardware/software package, together with the computers, cables and connectors, shared resource managers (servers), and specialized application software function in three major configurations: as small area interconnect systems, as a tributary to a larger network, and as a peripheral bus between Apple computers and their dedicated peripheral devices. This chapter describes AppleTalk to provide the external developer with a coherent picture of the firmware involved.

Firmware RAM Memory Map

The following depicts the firmware RAM map, used as a receive and write buffer:



Pointers, ID Bytes, and Entry Points

The following flags and pointers are set up in slot 7, in Cortland's ROM starting at location $C700.

Address	Purpose
$C705	$38 Identifier byte #1
$C707	$18 Identifier byte #@
$C70B	$01 Generic signature byte
$70C	$9B Device signature byte 9 = Network or bus interface card/firmware B = Apple Tech Support ID nibble
$C700	$xx Offset to Pascal error routine
$C70E	$xx Offset to Pascal error routine
$C70F	$xx Offset to Pascal error routine
$C710	$xx Offset to Pascal error routine
$C711	$88 Non-zero indicates no offsets follow
$C712	---- APPLETALK entry point ----
$C715	---- REBOOTAPTALK entry point ----
$C718-$C7FD	Reserved as code area
$C7FF	$00 RELVERNUM release version number
$E0C038	SCCADATA register
$E0C039	SCCAREG register
$E0C03A	SCCBDATA register
$E0C03B	SCCBREG register
$E0C0xx	Enable 1/4-second timer interrupt
$E0C0xx	1/4-second timer status
$bb047F	User sets to $Cn ($C7 for Cortland) to indicate a printer driver is installed.
$bb06FF	Printer driver entry point bank address
$bb077F	Printer driver entry point low byte of address-1
$bb07FF	Printer driver entry point high byte of address-1

bb = $00 if shadowing is on.
= $E0 if shadowing is off.

Note: At Reset time:

1. All SCC registers and functions are reset. This also turns off SCC interrupts and the SCC's ability to interrupt.

2. All buffer pointers and variables used by AppleTalk are reset.

3. The timer interrupt capability in the Mega II that AppleTalk uses is disabled.

Booting

This section describes AppleTalk booting , frame definitions, and the booting sequence.

General Information

Cortland AppleTalk can be booted in three ways:

1. The MENU program options to start up from internal slot 7 have been chosen.
2. The user types in IN#7 or CALL 50965 from BASIC.
3. The user types in $C715G from the Monitor or JMPs or JSRs to $C715 from a program.

The following sequence of events occurs during booting:

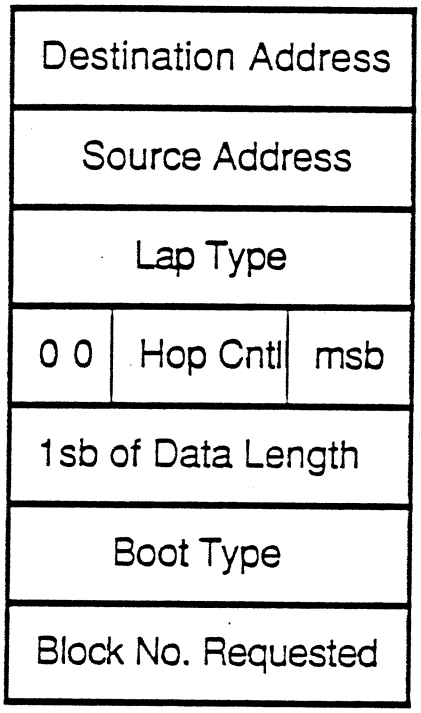
1. A series of transfers between the AppleTalk firmware and main system RAM occurs. The higher-level protocol, necessary to request boot information from the master station, is being moved from Cortland ROM to system RAM for execution. The boot code is placed at $200 to $3F0 and uses text page 1 ($400-$7FF) as a display/data buffer using $200 as the execution address. This allows all memory from $800-$BFFF to be used for storing the main boot program loaded from the master station.
2. When the transfers are complete, the AppleTalk boot code jumps to $200.
3. The RAM code establishes communications with the master/teacher station and requests the main boot code. The boot code could be ProDOS or Pascal or whatever. When the boot code is loaded, the RAM code causes the boot code to begin execution.
4. The slave station is a fully operational system that accesses files, at the master station, and a print station via AppleTalk (assuming FAP and PAP have been loaded with the operating system). The slaves cannot communicate between themselves.

Boot Sequence Frames

The following frames are used for normal boot sequences:

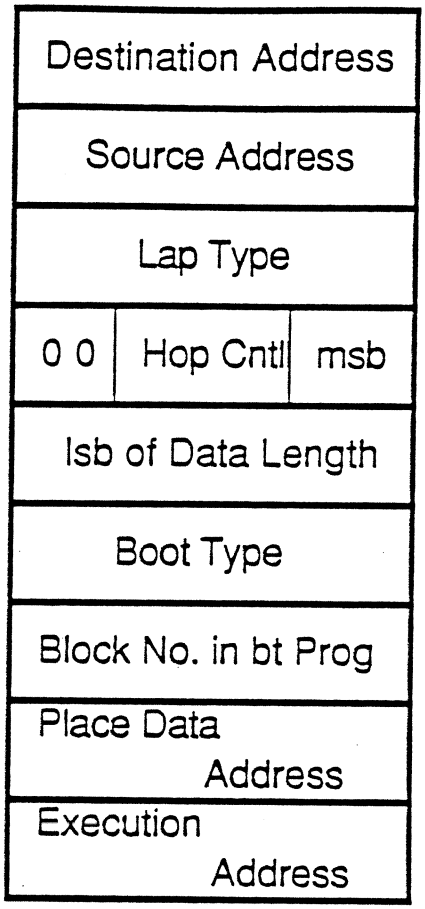
Boot Request Frame

The boot request frame is used by the slave station to request boot information, such as all boot blocks or specific boot blocks.



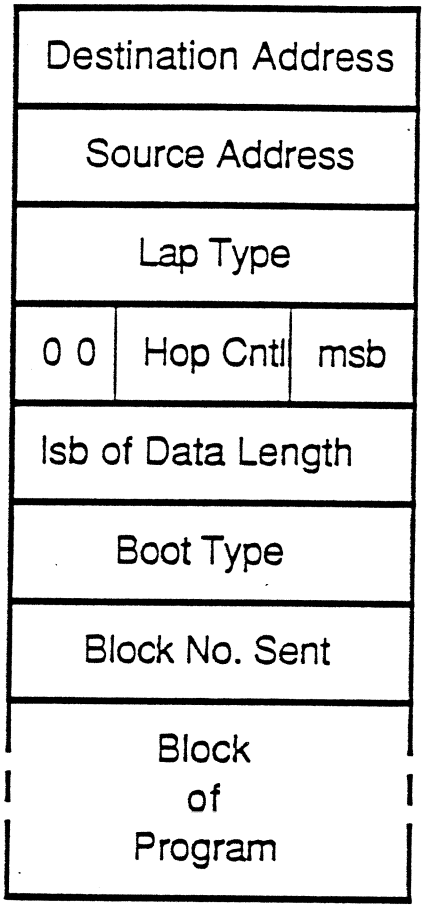
Boot Information Response Frame

The boot information frame is sent to the slave station by the master to inform the slave station of the boot program it is about to receive.



Boot Response Frame

The boot response frame is used by the master station to reply to the slave station with specific boot blocks.



Bytes Within Frames

The destination address for the Boot Request Frame is $FF. A station coming on-line doesn't know the master station's number.

The sending station's address number is the source address.

Lap Type is $0B for all boot transaction sequences.

The msb is the most significant two bits of the data length in the packet. Packet data length includes all bytes except the destination address, source address, and lap type.

The lsb Data Length is the least significant eight bits of the data length in the packet. Packet data length includes all bytes except the destination address, source address, and lap type.

The following describes boot types:

0 = Request for boot information
1 = Send boot blocks request

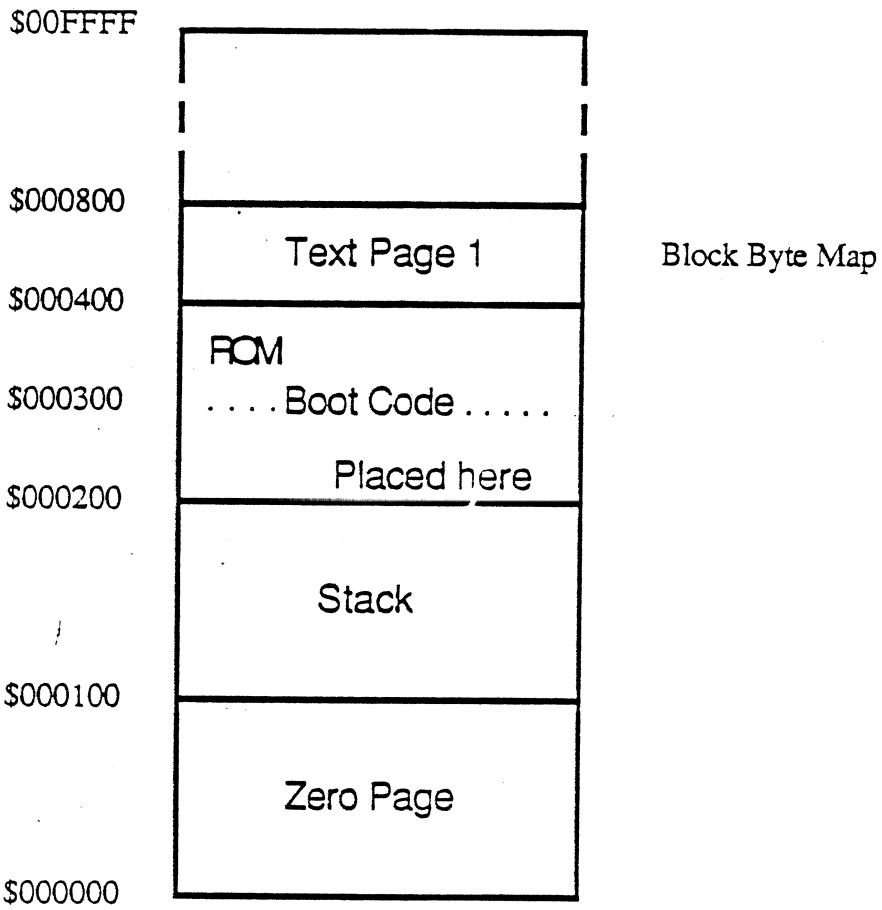
2 = Send specified boot block request
$80 = Boot information frame
$81 = Specific boot block

Block numbers range from 0 to $FF and consist of 512 bytes.

The place-data address is the starting address where the slave station places the main boot program as it receives it from the master station.

The execution address is the address to which the boot program should jump to start the main boot program.

Boot Routine Memory Map



The ROM boot code is placed at $00200 by the firmware after the user initiates a boot sequence.

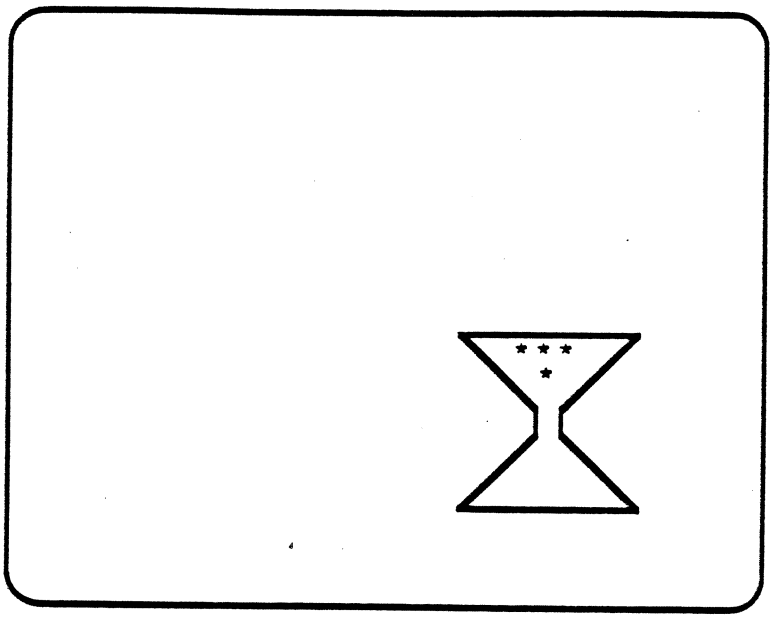
Text page 1 is the byte map for the boot program as it is being transferred from the master station to the slave station.

Locations $00-$1F and $56-$FF are used by the ROM's boot program as it loads the boot program from the master station.

A '.' will appear on the screen to correspond to a block number which is to be loaded from the master station.

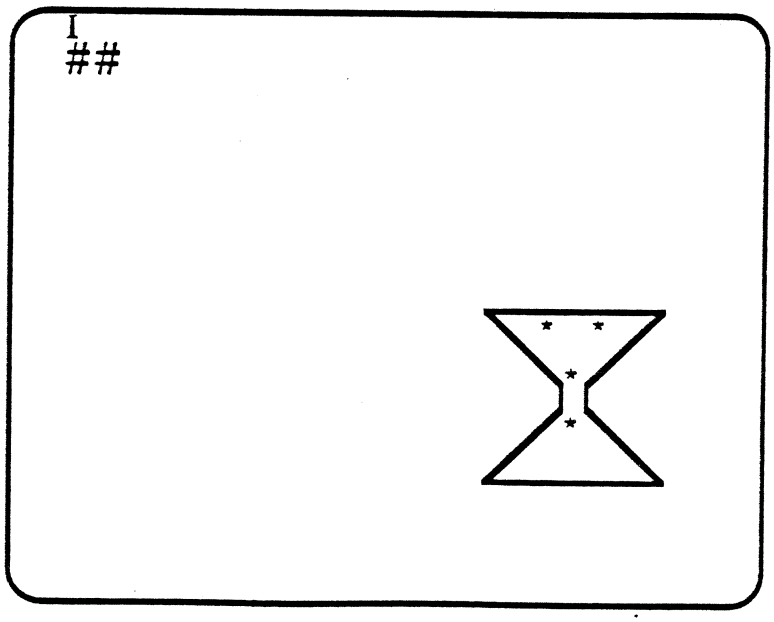
Slave Boot Screens

Initial Screen



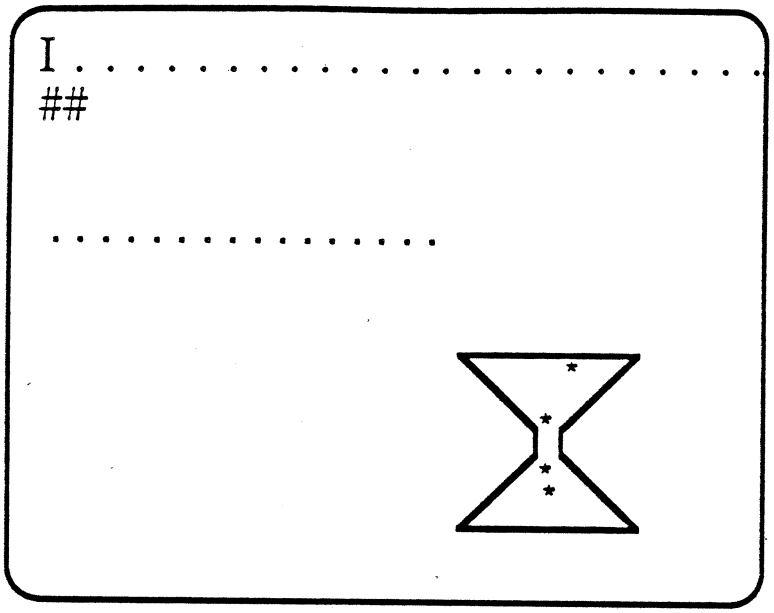
After a station # (node number) is determined, the following screen appears. The ##, in the upper-left corner, is the node number in hexadecimal.

Second Screen



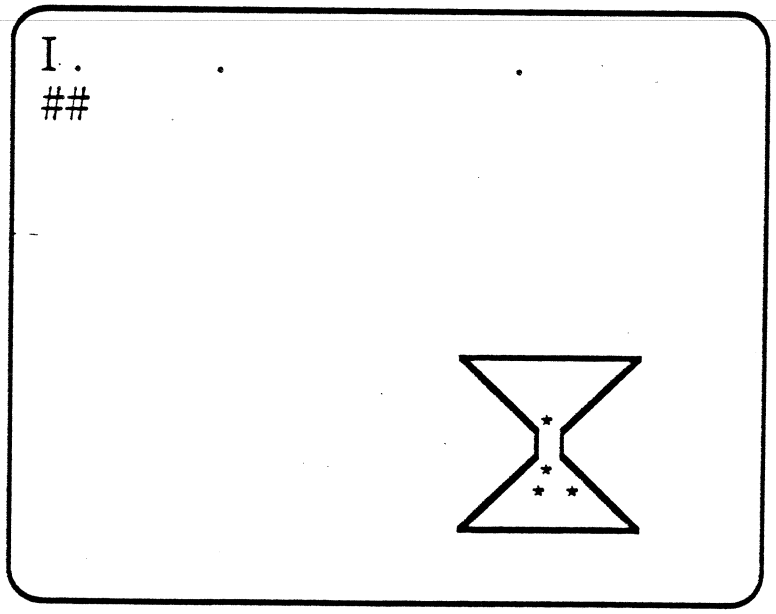
After the boot information frame is received, the following frame appears:

Third Screen



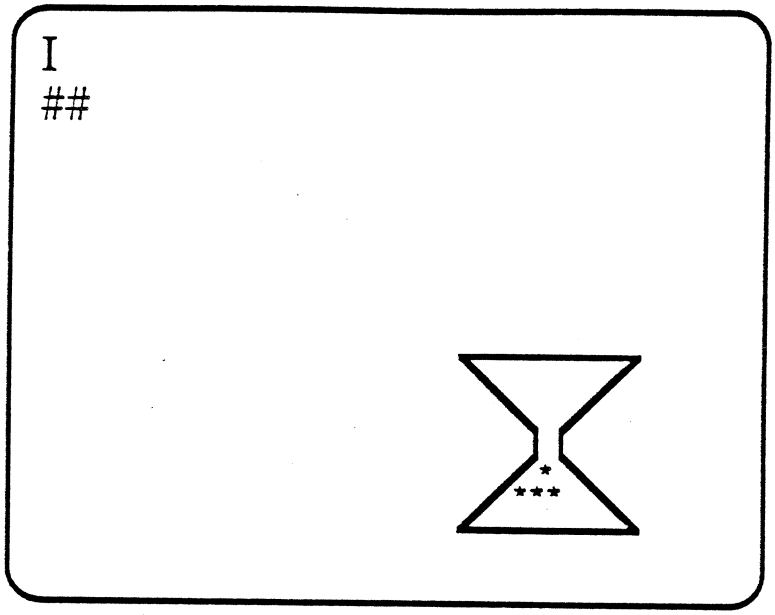
After timeout occurs, or after block 1 (blocks are received in reverse order) is received, the next screen appears. The dots left on the screen may or may not appear. They indicate unreceived blocks which are to be requested one at a time after this screen appears.

Fourth Screen



The final screen appears only after all blocks required have been received. Take note that all the 'grains' of 'sand' are now at the bottom of the hour glass.

Final Screen



The 'I' appearing in the Program Screen and the Byte Map Screen represents an indicator that the program is still running. It increments every 1/4 second until the entire user boot program is received and the firmware's boot program jumps to the starting address of the user's boot program.

Boot Sequence

1. Power up master station.
2. Initiate the boot sequence on the slave station.
3. The slave station broadcasts a Boot Request Frame with a boot-type 0 to get the Boot Information Frame. It broadcasts it every 1/4 second until the master station responds.
4. The master station sends packets (blocks) sequentially one time only.
5. The slave station sends a directed packet to the master station asking for all boot frames (boot type=1).
6. The master station sends packets (blocks) sequentially one time only.
7. The slave station receives frames and places them in sequential order in memory according their block numbers.
8. The slave station determines which blocks it missed.

9. The slave station requests block numbers of frames it missed, one at a time, waiting 150 ms between requests.

10. The master station sends requested blocks to the slave station.

11. The slave initializes the AppleTalk firmware.

12. The slave station JMPs to the execution address.

13. The program, just loaded, takes control of the slave station.

Cortland User Interface

This interface requires that user RAM is free of the ATLAP code. It is implemented to ensure an identical interface between the Apple II and Cortland. This interface allows the user to write different higher-level protocols (such as a new DDP) and still be able to use our LAP protocol. This generic LAP protocol interface allows us to enhance and improve the LAP software and hardware without requiring changes to the application-writer programs. The firmware entry points are in a fixed location in the $Cn00 ($700 in Cortland) space that is compatible with Apple II.

User Interface

The DDP accesses the LAP in the following way:

This interface requires only one entry point into the $Cn00 space. Future maintainability is simple because we need only to ensure that the AppleTalk entry point is maintained.

AppleTalk Call

```
LDY #<PARAMLST      ;Y = hi byte of parameter list address
LDX #>PARAMLST      ;X = lo byte of parameter list address
LDA #$Cn            ;A = the slot # of the AppleTalk interface+$C0
                      ($C7 in Cortland)
JSR APPLETALK       ;Call the interface (in Apple II ROM/RAM and in
                      Cortland)

BNE ERRROUTINE      ;<>0 then an error occurred
```

Note: Decimal mode will always be clear upon exit from the AppleTalk routines.

AppleTalk PARAMLST

```
DFB #COMMANDNUM     ;Function requested
                    -- All Command Calls --
                    $01 = INIT
                          Initialize the interface
                    $02 = READREST
                          Read rest of buffer
                    $03 = WRITE
```

```
                    Write a buffer
              $04 = STATUS
                    Check if AppleTalk interrupted Set/Reset
                    interrupt masks
              $05 = READPROT
                    Read protocol from buffer
DW/DFB        ;Data pointers/actual data to pass to/from AppleTalk
               buffer
```

PARAMLSTs for Each Call

INIT Command Number 1

```
DFB $1        ;Command number for INIT call.
DS 1,0        ;Misc information to pass to the AppleTalk firmware
                1. $00, then normal init.
                2. $FF, then find new node address using a
                   random number and do normal init.
                3. $xx if 1 to $FE (1 to 254), then find new
                   node address but use $xx as starting
                   address when determining a new station
                   address.
               Note: $01-$7F (1-127) are valid node ID
               addresses. $80-$FE (128-254) are used for
               servers only. This $xx option therefore lets
               you set up Cortland as either a normal node
               or a server node.
                4. Returns AppleTalk station address.
```

READREST Command Number 2

```
DFB $1          ;Command number for READREST call.
DW BUFFADDR ;   ;Address in user's program to hold the rest of the
                 data packet.
                      1. Address of read buffer (buffer to which
                         packet is transferred).
DS 1,0          ;Misc information to pass to the AppleTalk firmware
                      1. =0, then read rest of the data from the
                         AppleTalk firmware RAM buffer.
                      2. <> 0, then purge and don't read current
                            packet to be transferred.
DS 2,0          ;Number of bytes read during READREST call.
```

WRITE Command Number 3

```
DFB $3          ;Command number for WRITE call.
DW WRITETBL     ;Address in 6502 of pointer table containing
                 data to transmit.
                      1. Address of write buffer pointer.
WRITETBL EQU *                          ;Generic form
           DW NUMDATABYTES              ;Number of bytes to read
           DW DATABUFFER                ;Pointer to data buffer
           DW NUMDATABYTES2             ;Number of bytes to read
```

```
DW DATABUFFER2          ;Pointer to data buffer
        •
        •
        •
DW $FFxx                ;Pointer table terminator
```

Sample WRITETBL (DESTADR, SRCADR, LAPTYPE need not be separated as this example shows).

```
WRITETBL EQU *
        DW $0001        ;Number of bytes
        DW DESTADR      ;Pointer to destination address
        DW $0001        ;Number of bytes
        DW SRCADR       ;Pointer to source address
        DW $0001        ;Number of bytes
        DW LAPTYPE      ;Pointer to LAP type
        DW DDPLEN       ;Number of bytes
        DW DDPBUF       ;Pointer to DDP data
        DW ATPLEN       ;Number of bytes
        DW ATPBUF       ;Pointer to ATP data
        DW MISCLEN      ;Number of bytes
        DW MISCBUF      ;Pointer to misc data
        DW $FFxx        ;Pointer table terminator
```

STATUS Command Number 4

```
DFB $4      ;Command number for STATUS call.
DS 1,0      ;Misc information to/from the AppleTalk firmware.
             This parameter byte is explained below.
```

The STATUS call sets interrupt masks and returns interrupt status to the user. If STATUS is called with a parameter byte of -, then the call sets the interrupt masks only. If the parameter byte is +, then the call is requesting interrupt information.

B7	B6	B5	B4	B3	B2	B1	B0

A '-' parameter byte is defined as follows:

B7 = 0	Return interrupt status request.
B7 = 1	Set interrupt mask request.
B6 = 0/1	Enable/disable 1/4-sec timer interrupt.
B5 = 0/1	Enable/disable packet ready interrupt.
B4-B0	Reserved

A'+' parameter byte is defined as follows:

B7 = 0	Return interrupt status request.
B6-B0	Reserved

Above call returns with parameter byte defined as follows:

B7 = 0/1	AppleTalk packet or/and timer event occurred.
B6 = 0/1	1/4-sec. timer went off.
B5-B4	Reserved
B3-B0	1 bit set for each packet in buffer (1 packet maximum in Cortland).

READPROT Command Number 5

```
DFB $5          ;Command number for READPROT call.

DW BUFFADDR     ;Address in user's program in which part of data
                packet is stored.
                    Address of read buffer (buffer to which
                    packet is transferred).

DS 2,0          ;Number of bytes
                    Number of bytes to read.
```

Notes:

1. READPROT can read from last position+1 accessed. It cannot read data prior to the last read data postition in the current packet.

2. For all calls, carry will return SET if an error occurred; the accumulator will contain the error code.

3. For a STATUS call, carry will return SET (indicating the user was wrong in assuming that AppleTalk was the interrupting device). If AppleTalk was the interrupting device, carry will return CLEAR (indicating AppleTalk was the interrupting device).

Error Codes

Command error = $FF for any call where the command # does not equal 1, 2, 3, 4, or 5.

INIT call errors:

4 = Could not get unique AppleTalk address for station or in the Apple II version. Could not talk to the Apple II AppleTalk protocol converter box.

READPROT call errors:

1 = No packets in buffer to read.
2 = Multipurpose buffer overflowed (not possible in Cortland).
3 = Tried to read past end of current data packet.

READREST call errors:

1 = No packets in buffer to read.
2 = Multipurpose buffer overflowed (not possible in Cortland).

WRITE call errors:

5 = Number of bytes to send >603.
6 = Number of bytes <3.
7 = Excessive deferrals.
8 = Too many collisions.
9 = Illegal lap type <>127 ($7F not allowed).

STATUS request call errors:

$A = AppleTalk was not the interrupting device.

STATUS set interrutp mask call errors:

None possible.

Description of Calls

INIT: Start timer. Inhibits all AppleTalk interrupts and resets AppleTalk IRQ sources.

Note: STATUS must be called with an interrupt mask to enable AppleTalk interrupts to be returned.

INIT call returns:

C = 0 if no error occurred.
C = 1 if an error occurred.
A = Error code.
X/Y/V = Scrambled.

READPROT: Called to read xx number of bytes from the buffer beginning with the last read byte+1 in the buffer. This call is used by the different protocol layers to read their headers from the multi-purpose buffer into their buffer.

The READPROT call returns:

C = 0 if no errors occured.
C = 1 if an error occurred.
A = Error code.
X/Y/V = Scrambled.

Note: READPROT can read from last position+1 accessed. It cannot read data prior to the last read-data position in the current packet.

READREST: Reads from last position+1 accessed (via READPROT), or from the start of packet if no previous READPROT was called, and places data in user-specified buffer. Allows user to purge the current packet without reading it if desired.

The READREST call returns:
C = 0 if no errors occured.
C = 1 if an error occurred.
A = Error code.
X/Y/V = Scrambled.

WRITE: Called by the appropriate protocol level to move data from the protocol buffer and send a datagram on AppleTalk. WRITE passes a pointer to a table of pointers and byte counts that include sequentially, a correct data packet with all protocols intact and data present. This table is built by each protocol above the LAP including its protocol data in the correct sequence in a common table found in the DDP.

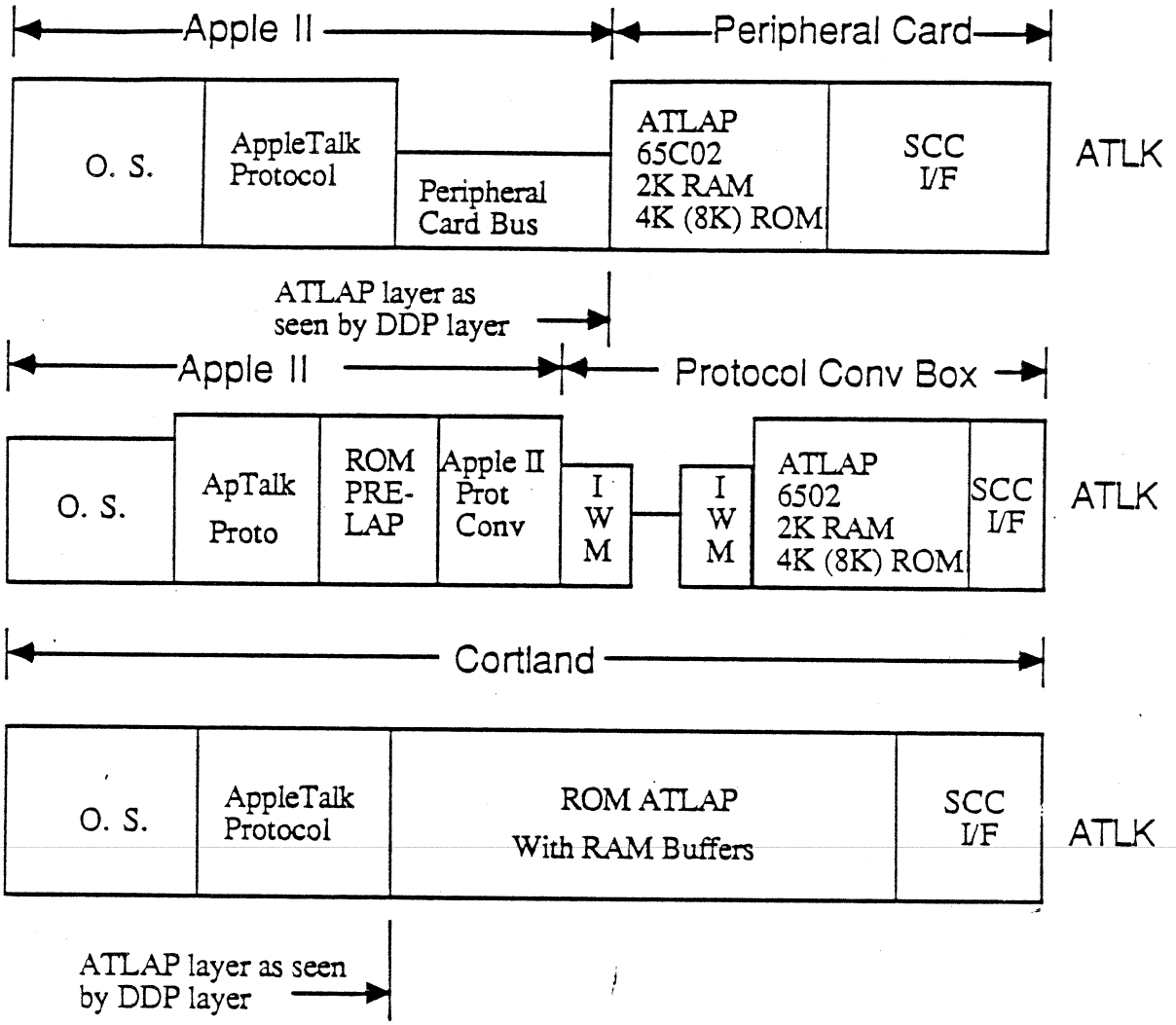
Note: The source node number is placed over the second byte in the packet to be written by the AppleTalk firmware. Therefore, you don't need to know your station (node) number to transmit a packet. You must, however, provide space for the source address to go when defining a packet.

The WRITE call returns:
C = 0 if no errors occured.
C = 1 if an error occurred.
A = Error code.
X/Y/V = Scrambled.

STATUS: Called when an interrupt occurs to determine if AppleTalk was the interrupting source. If C=0, it was; if C=1, it was not. If AppleTalk was the interrupting device, STATUS returns whether it was a 1/4-second timer interrupt, or a packet-ready interrupt. If an AppleTalk source was not the interrupting device, the accumulator register returns $A as the error code. STATUS is also called to set the interrupt masks. In every case, whether the interrupt mask allows interrupts or not, the STATUS call parameter byte will return the current status of the events which have taken place relating to AppleTalk. This allows Cortland's AppleTalk ability to be used in a polling mode if for some reason the user decided not to use our higher-level protocols (our higher-level protocols require the use of interrupts) and wrote ones not requiring interrupts.

The STATUS call returns:
C = 0 if AppleTalk was the interrupting device (clears interrupt).
C = 1 if AppleTalk was not the interrupting device.
A = Error code.
X/Y/V = Scrambled.

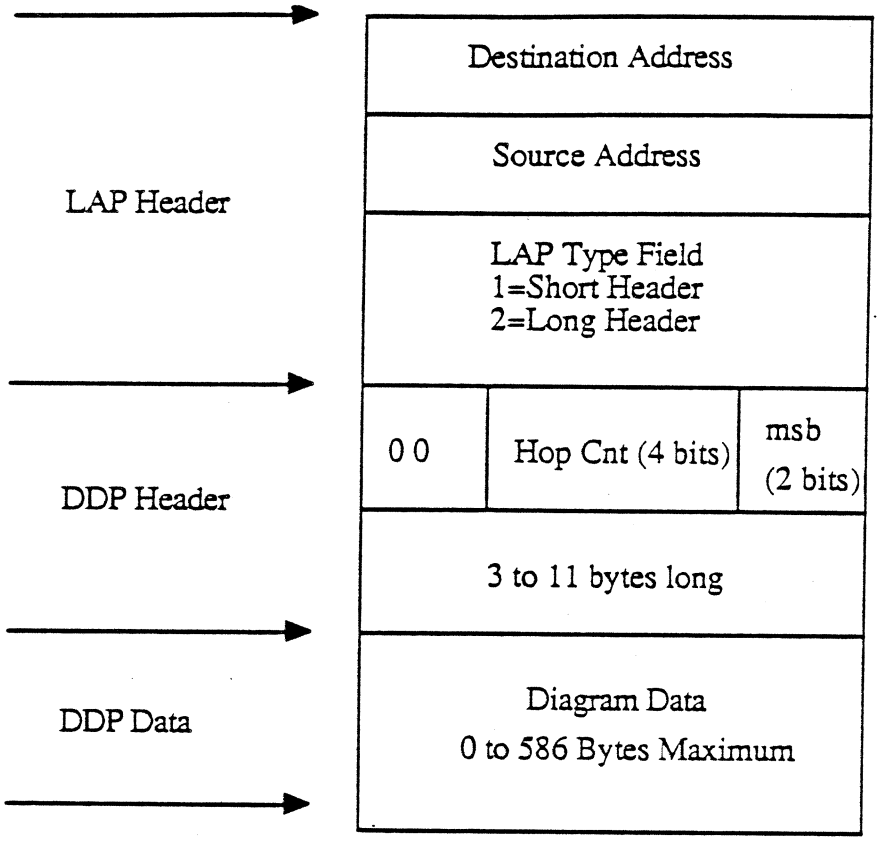
Apple II AppleTalk Interface General Diagram



Receive Buffer

During an interrupt to the 65816, the firmware interrupt handler will determine if it is an AppleTalk-related interrupt. If it is, it calls AppleTalk firmware to handle the interrupt, read data into the receive buffer, and call the user if required. When the user is interrupted, he will call theSTATUS routine to determine the type of AppleTalk interrupt that occurred (a packet ready to read or a 1/4-second timer interrupt). If a read is required, the user first calls READPROT, which enables the DDP to determine which node the message is for. That particular node will call READREST, which will read the rest of the data packet. If no packet is in the buffer when READPROT or READREST is called, the user will receive a no-packets-available error.

Receive Buffer Packet Data Structure



Packet Rejection Error Conditions

The firmware automatically rejects an incoming packet under the following conditions:

- Any SCC error.
- More than 603 bytes are in the incoming packet
- The number of bytes-3 received do not equal the length byte.
- No characters received within 1 character time (approximately 34.722 microseconds.)
- A WRITE operation is in progress.

In every case, the operation is not interrupted if any of the above conditions occur. The firmware will reset its pointers and wait for more packets to be sent.

Interrupting The User

The AppleTalk firmware interrupts the user when it has received a datagram the user should know about or when 1/4-second has elapsed. The timing interrupt, like the SCC, cannot directly interrupt the user for any reason (It interrupts the 65816, but it is not passed to the user unless requested). The AppleTalk firmware controls the user interrupt. During the interrupt routine, a call to STATUS will inform the user what type of interrupt occurred. If the interrupt was from AppleTalk, carry = 0; if not, carry = 1.

The ability to interrupt the user is determined by the interrupt mask sent to AppleTalk firmware during the last STATUS call. The mask can be set to allow timer interrupts and/or packet-ready interrupts in any combination.

It is possible (although not with our higher-level drivers) to use AppleTalk in a non-user interrupt mode by polling the AppleTalk firmware. This is accomplished by periodically performing a STATUS call, ignoring the carry bit, and decoding the status byte.

- Bit 7 is set when an AppleTalk event occurred.
- Bit 6 is set if the 1/4-second timer lapsed.
- Bit 0 sets to indicate a packet was received since the last READREST call.

Using the above data, the user can call READPROT and READREST to extract the packet data from AppleTalk's firmware RAM buffer.

> Note: For Cortland's AppleTalk to work, interrupts must be enabled whether the user wants to be interrupted or not. If the user doesn't want to be interrupted, the firmware will trap, decode, and act on all AppleTalk interrupt sources transparent to the user.

Resetting Firmware and Hardware

AppleTalk firmware and hardware can be reset in three ways:

1. Press CONTROL-RESET.
2. Press OPEN-APPLE-CONTROL-RESET.
3. Power up the system.

lapENQ, lapACK, lapRTS, lapCTS

LAP enquiry, acknowledge, request to send, and clear to send will be handled transparently to the user. The AppleTalk firmware will process and respond when these frames occur or should occur.

AppleTalk firmware has recognizable ID bytes for ProDOS and Pascal. Apple II AppleTalk uses the generic Pascal 1.1 firmware entry points, however, AppleTalk does not support any Pascal generic firmware calls directly, nor does it support any Pascal 1.0 firmware entry points. A machine-language driver must be written for Pascal and ProDOS for these operating systems to access AppleTalk.

AppleTalk ProDOS drivers reside in the main language card, bank 2, at locations $D400-$DFFF. The AppleTalk driver for Pascal resides on the heap.

Printer Hooks Via AppleTalk Firmware

AppleTalk firmware does not provide all the protocol and routines necessary to output to a print server. However, by providing proper hooks in the AppleTalk interface firmware, you can output to a printer driver located in Apple II's main memory. This allows BASIC and ProDOS application programs to access the AppleTalk interface firmware as if it were a normal printer card. Entry at $Cn00 is for an initialization call for the printer driver, entry at $Cn05 is for inputting a character, and entry at $Cn07 is for outputting a character to the printer.

Entry at $Cn00 is to initialize the printer driver interface, if one is loaded into main memory. To determine if a driver is available, perform the following step:

> Test the first screen hole, $47F, to verify that it is $C7 ($C7 is the flag which indicates that a driver has been installed).

If a driver is not available, the Monitor ROM is mapped in and a JMP to the Monitor RESET routine is executed.

If a driver is available, the AppleTalk interface firmware performs the following:

1. Loads the printer driver address-1 low byte from screen hole location $77F and pushes it on the stack.

2. Loads the printer-driver address-1 high byte from screen hole $7FF and pushes it on the stack.

3. Loads the printer driver bank address from screen hole $6FF and pushes it on the stack.

4. Performs an RTS which goes to the driver if shadowing is on; performs an RTL which goes to the driver if shadowing is off.

The following depicts the information AppleTalk interface firmware passes to the printer driver:

Y = user Y
X = user X
A = user A
P = Print character status:
V=1 if init printer driver requested
C=1 if input to printer
C=0 if output to printer

It is assumed that part of the printer-driver initialization code will be to place $Cn at screen hole location $47F and its execution address-1 into screen holes $77F (low byte), $77F (high byte), and $6FF (bank byte).

Appendix B

Commands to uC

Init commands are to be two-byte commands.
Bit 76543210

00000000 -
00000001 ABORT COMMAND
00000010 RESET KEYBOARD uC
00000011 FLUSH KEYBOARD
00000100 SET MODES using next byte as in Table B-1.
00000101 CLR MODES using next byte as in Table B-1.

00000110 * SET CONFIGURATION BYTES using next 3 bytes as follows:
Byte 1:
HI nibble: FDB mouse address
LO nibble: FDB keyboard address
Byte 2:
HI nibble: Character set (needed for certain languages)
msb set if keypad '.' swapped with ','
LO nibble: Set keyboard layout language as in Table B-2

Byte 3:
HI nibble: Set delay to repeat rate (3 bits)
0: 1/4 second
1: 1/2 second
2: 3/4 second
3: 1 second
4: NO REPEAT

LO nibble: Set auto-repeat rate (3 bits)

0: 40 keys/second
1: 30 keys/second
2: 24 keys/second
3: 20 keys/second
4: 15 keys/second
5: 11 keys/second
6: 8 keys/second
7: 4 keys/second

00000111 SYNCH COMMAND
Sets MODES byte (See command 4 or 5 above) followed by configuration bytes (command 6). This command is issued by the system after a keyboard reset. After receiving the command, the uC resets itself back to its internal power-up state, and then resets FDB devices.

00001000 WRITE uC MEMORY
Send 1-byte address (for RAM) following by 1 byte of data.

00001001 READ uC MEMORY
Send 2-byte address of uC location (ROM or RAM).
00001010 READ MODES BYTE (See command 4 or 5 above)
00001011 * READ CONFIGURATION BYTES (Returned in data latch)
Note: Returned in reverse order from command 6 above.
Byte 1:
HI nibble: FDB mouse address
LO nibble: FDB keyboard address
Byte 2:
HI nibble: Character set (needed for certain languages)
LO nibble: Set keyboard layout language
Byte 3:
HI nibble: Set delay to repeat rate (3 bits)
LO nibble: Set auto-repeat rate (3 bits)
00001100 READ THEN CLEAR FDB ERROR BYTE (returned in data latch)
00001101 GET VERSION NUMBER (returned in data latch)
Also, returns port R, which is an undefined input port on uC; in HI nibble.
00001110 READ CHARACTER SETS AVAILABLE
Returns number of bytes, then the data. This command is used by Control Panel to determine which character sets are available in the system. This assumes that each uC is paired with a specific mega chip. (However, mega chips may be paired with more than one uC). The order that the character sets are returned is important. The first number returned corresponds to character set 0 in the mega chip, while the next number is character set 1.
00001111 READ LAYOUTS AVAILABLE
Returns number of bytes, then the data. This command is used by the Control Panel to determine which keyboard layouts are available in the sysem. Again, like the character-sets-available command, the order in which the numbers are returned is important. The first number returned represents layout 0 in the uC. A predefined table defines which number corresponds to which layout language. The following commands will be added, however, the exact protocol has not been determined:
00010000 RESET THE SYSTEM
Pulls the reset line low for 4 ms.
00010001 SEND FDB KEYCODE
Pretend that the second byte is the FDB keycode. This command can be used to emulate an FDB keyboard, by accepting keycodes from a device and then sending them to the uC to be processed as keystrokes. This command will not process either RESET-up or RESET-down codes; therefore, they must be trapped out before using this command. This command can be used to watch for key up sequences.
0001---1 - -
001----- - -
01000000 RESET FDB
Pulls the FDB low for 4 ms. Care must be taken with this command because resetting an FDB keyboard will clear any pending commands including all key-up events. This means that if a keystroke is used to launch this command while the key is released, the key-up code will be lost and the key will auto-repeat until another key is pressed. All keys should be up before this command is executed.
01001000 RECEIVE BYTES
Command with address is in second byte. The system starts by sending a command byte on the FDB, then waits for the uC to pass back any data that it receives. The command returns bytes in opposite order (n->1).
01001num . TRANSMIT num BYTES
Command with address is in second byte. Note: If num = 0, then the command is RECEIVE BYTES described above. The system starts by sending a command followed by from 2 to 8 data bytes (num+1) to the uC, which is transmitted over the FDB. The

command sent will be transmitted directly as the FDB command byte, which is the first byte received after the TRANSMIT num BYTES command.

0101abcd ENABLE SRQ ON FDB DEVICE AT ADDRESS abcd
[Send command = abcd Listen R3 (abcd1011)]
[Data = 0010abcd 00000000]

0110abcd FLUSH BUFFER ON FDB DEVICE AT ADDRESS abcd
This command is dangerous--see RESET FDB description.
[Send command = abcd0001]

0111abcd DISABLE SRQ ON FDB DEVICE AT ADDRESS abcd
This command may be dangerous. If data is pending when this command is executed, then the pending data may be lost. For example, if SRQ is disabled on the FDB keyboard, then all key-up codes may be lost. See RESET FDB description.
[Send command = abcd Listen R3 (abcd1011)]
[data = 0000abcd 00000000]

1Cxyabcd Poll FDB device
Address: abcd
Register: xy
Command: 1C
C = 1, talk
C = 0, listen

This assumes that the FDB command is to either Talk or Listen. Other FDB commands are implemented using a 2-byte protocol (see above). If the command is Listen, then a 2-byte transfer is assumed.
[Send command = abcd1Cxy]
[data = 1st byte, 2nd byte (if Listen command)]

Returns bytes in opposite order than received (n->1).
Note: All commands that require more than a 1-byte transfer, will automatically timeout in 10 ms if there is no response except the SYNCH command that may require 20 ms to process the FDB address byte.

Appendix C

FDB Keycodes

Code	Key	Differences	Code	Key		Differences
0	A		48	TAB		
1	S		49	SPACE		
2	D		50	'		
3	F		51	DELETE		
4	H		52	RETURN		*ENTER
5	G		53	ESCAPE		*NA
6	Z		54	CONTROL		*NA
7	X		55	OPEN-APPLE		*COMMAND
8	C		56	SHIFT		
9	V		57	LOCK		
10	* INTERNATIONAL		58	SOLID-APPLE		*Option
11	B		59	LEFT ARROW		
12	Q		60	RIGHT ARROW		
13	W		61	DOWN ARROW		
14	E		62	UP ARROW		
15	R		63			
16	Y		64	DELETE	KEYPAD	*NA
17	T		65	.	KEYPAD	
18	1		66	RT ARROW(*)	KEYPAD	
19	2		67	*	KEYPAD	*NA
20	3		68	?	KEYPAD	*NA
21	4		69	+	KEYPAD	*NA
22	6		70	LFT ARROW(+)	KEYPAD	
23	5		71	ESCAPE	KEYPAD	
24	=		72	DN ARROW(,)	KEYPAD	
25	9		73	,	KEYPAD	*NA
26	7		74	SPACE	KEYPAD	*NA
27	-		75	/	KEYPAD	*NA
28	8		76	RETURN	KEYPAD	*ENTER
29	0		77	UP ARROW(/)	KEYPAD	
30	]		78	-	KEYPAD	
31	0		79	(	KEYPAD	*NA
32	U		80	)	KEYPAD	*NA
33	[		81		KEYPAD	
34	I		82	0	KEYPAD	
35	P		83	1	KEYPAD	
36	RETURN		84	2	KEYPAD	
37	L		85	3	KEYPAD	
38	J		86	4	KEYPAD	
39	'		87	5	KEYPAD	
40	K		88	6	KEYPAD	

Code	Key	Differences	Code	Key	Differences
41	;		89	7	KEYPAD
42	\		90		KEYPAD
43	,		91	8	KEYPAD
44	/		92	9	KEYPAD
45	N		93		KEYPAD
46	M		94		KEYPAD
47	.		95		KEYPAD

Code for RESET UP ($FFFF) and RESET DOWN ($7F7F). Other keypad codes (>95) are passed directly through to the keylatch.