Chapter 6

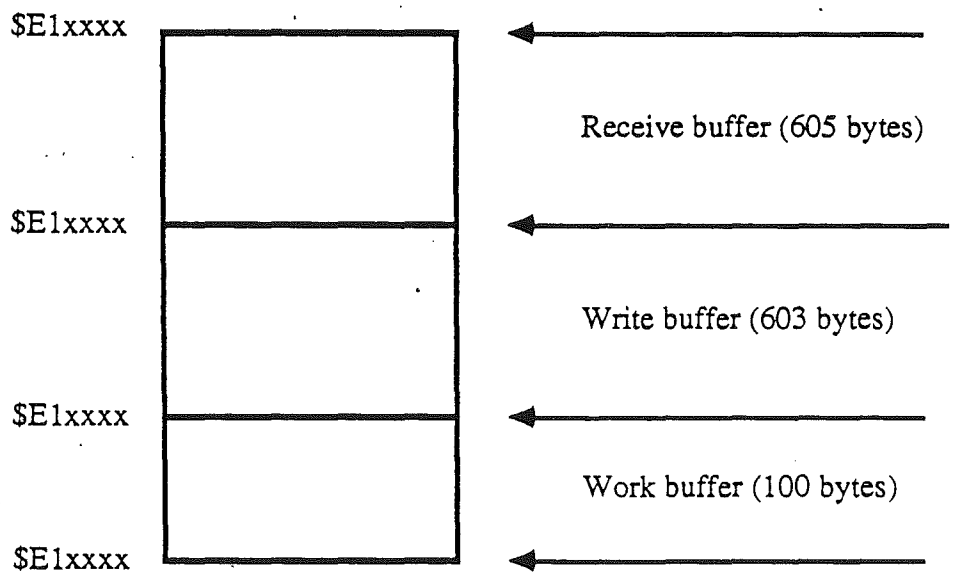
AppleTalk

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:

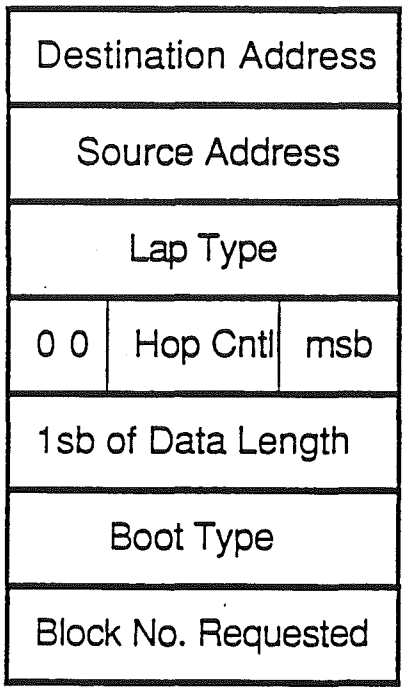
1. A series of transfers between the AppleTalk firmware and main system RAM occurs. The higher-level protocol, necessary to request boot information from the master station, is being moved from Cortland ROM to system RAM for execution. The boot code is placed at $200 to $3F0 and uses text page 1 ($400-$7FF) 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:

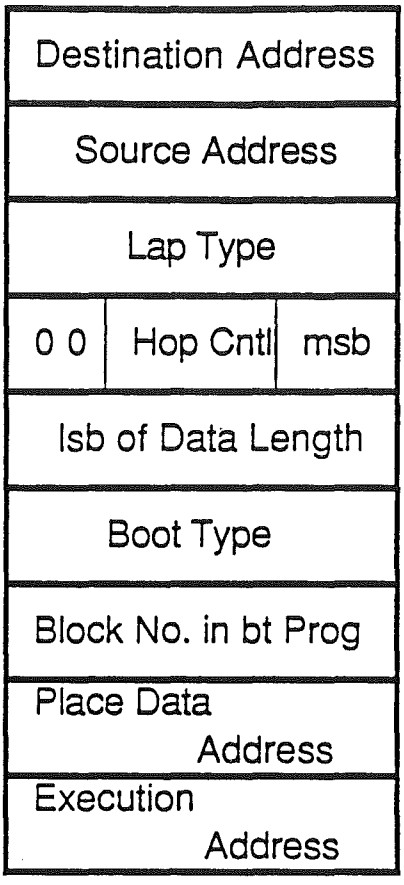
Boot Request Frame

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



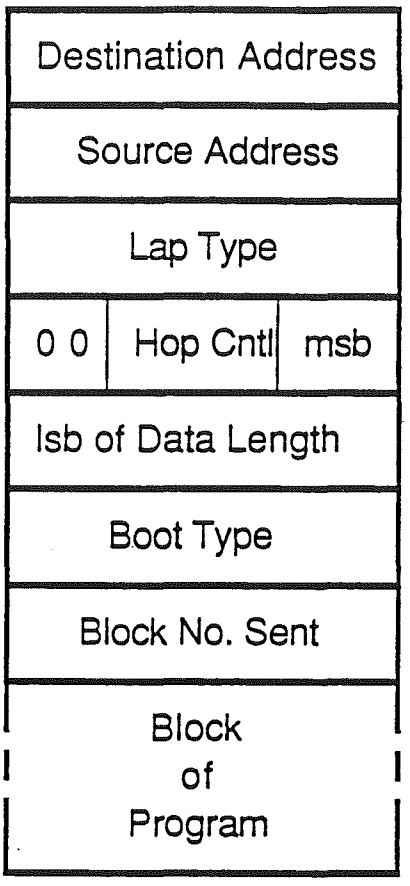
Boot Information Response Frame

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

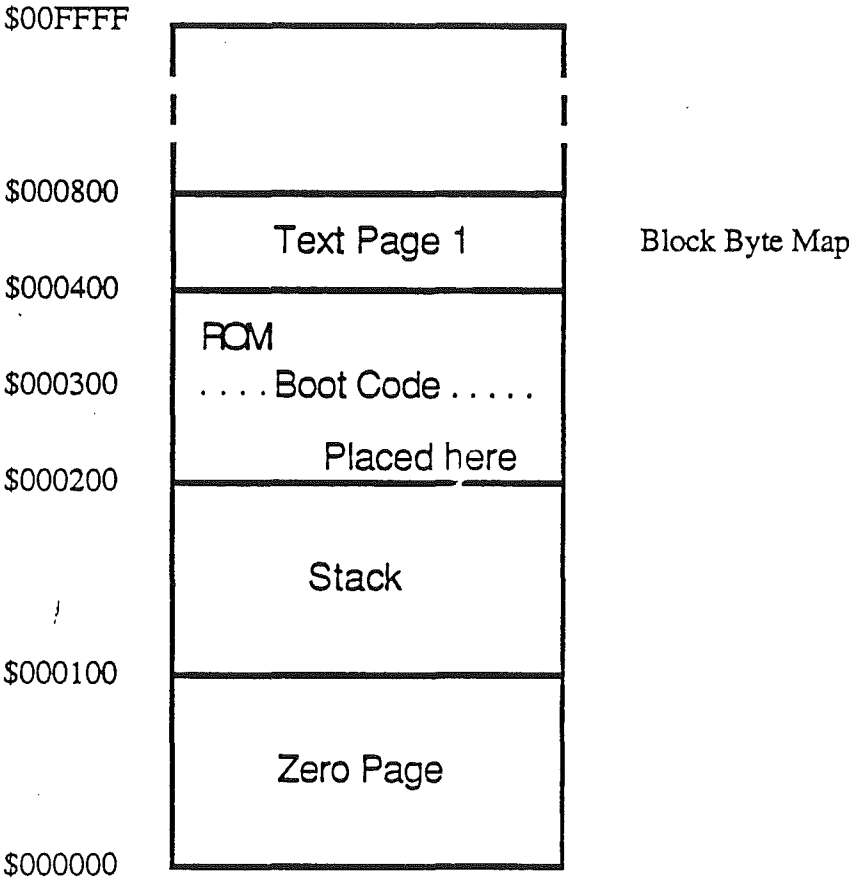
2 = Send specified boot block request
$80 = Boot information frame
$81 = Specific boot block

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.

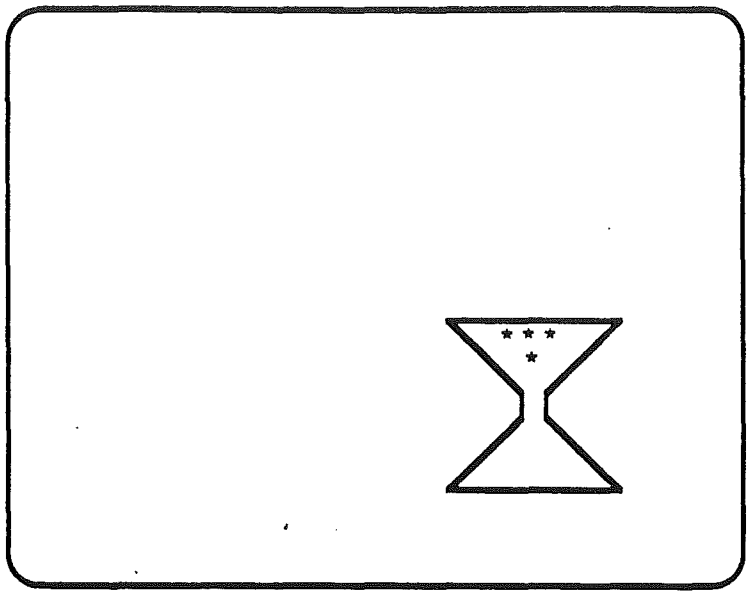
Text page 1 is the byte map for the boot program as it is being transferred from the master station to the slave station.

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.

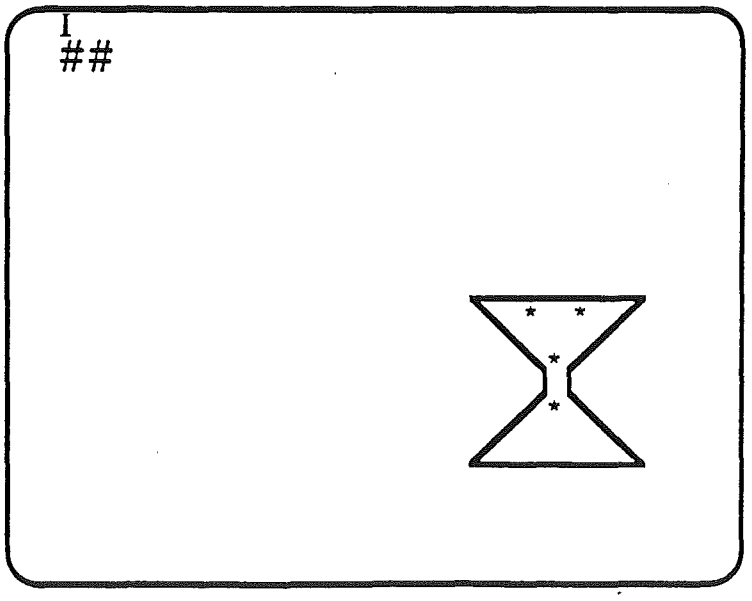
Slave Boot Screens

Initial Screen



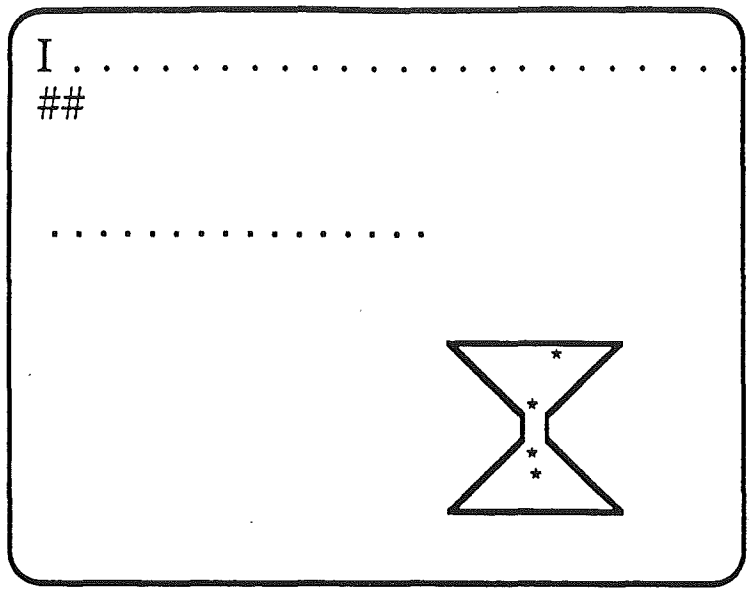
After a station # (node number) is determined, the following screen appears. The ##, in the upper-left corner, is the node number in hexadecimal.

Second Screen



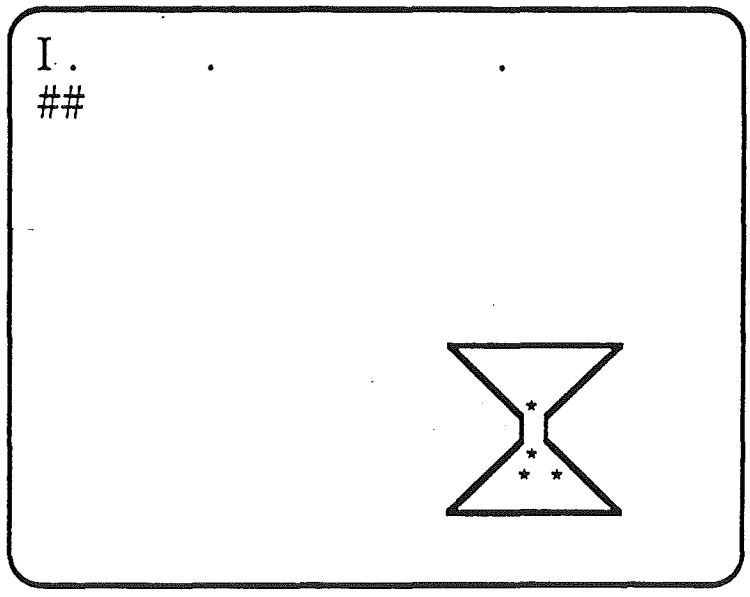
After the boot information frame is received, the following frame appears:

Third Screen



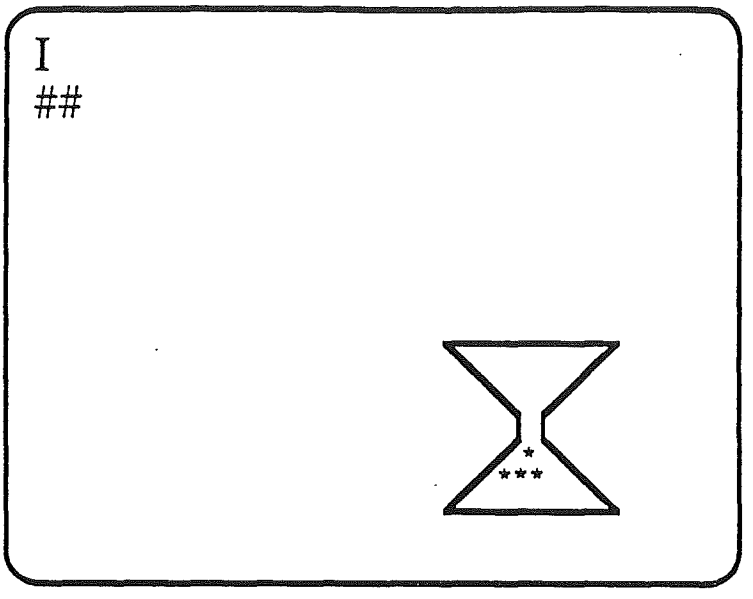
After timeout occurs, or after block 1 (blocks are received in reverse order) is received, the next screen appears. The dots left on the screen may or may not appear. They indicate unreceived blocks which are to be requested one at a time after this screen appears.

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.

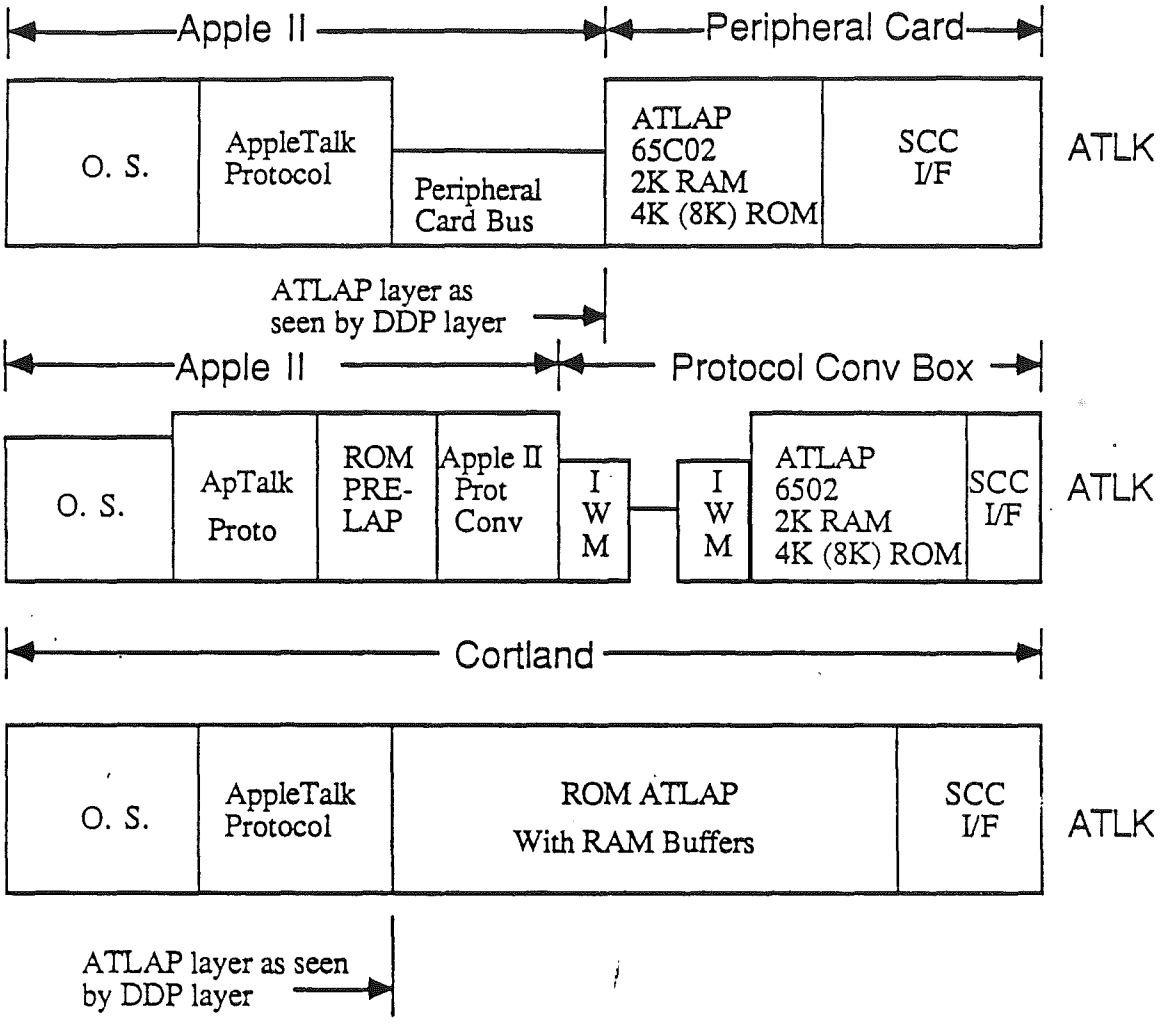
Note: The source node number is placed over the second byte in the packet to be written by the AppleTalk firmware. Therefore, you don't need to know your station (node) number to transmit a packet. You must, however, provide space for the source address to go when defining a packet.

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.

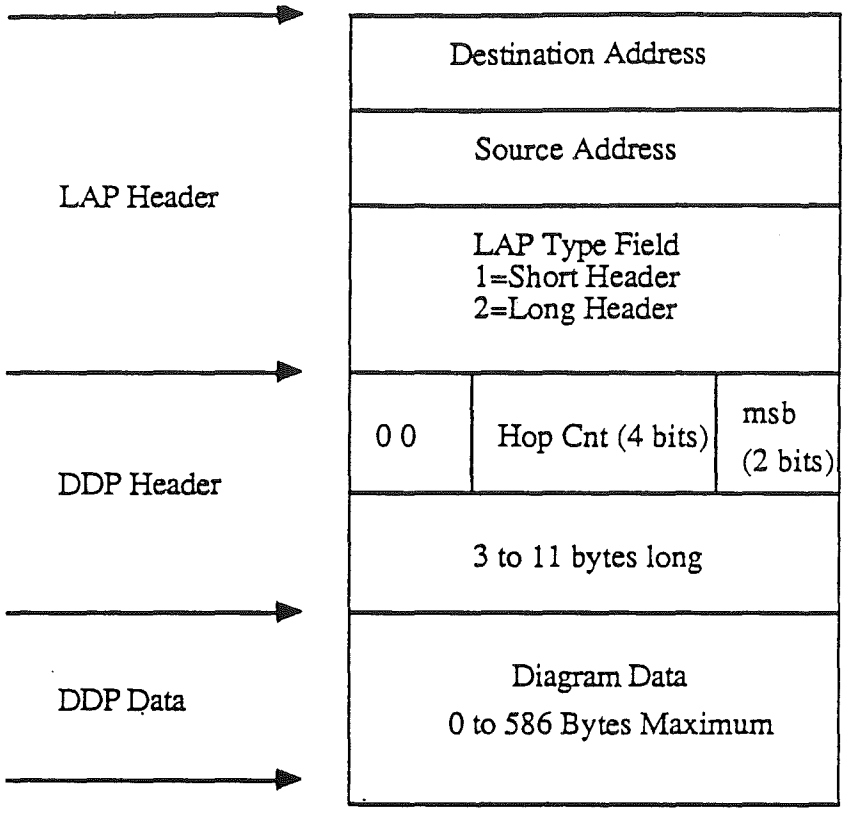
Apple II AppleTalk Interface General Diagram



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).