Date: June 12, 1985

Author: Fern Bachman

Subject: Columbia AppleTalk ERS

Document Version Number: 00:00

---

## General Information

AppleTalk on Columbia is significantly different from either the AppleTalk //e card or the AppleTalk //c box. Both of those systems had a dedicated 65C02 to handle AppleTalk communications with the SCC chip. In Columbia, the main system processor (65816) has to handle normal system operation and all necessary AppleTalk communications with the SCC chip. Without the dedicated 65C02, Columbia's SCC and timer interrupts must save and restore the operating environment even if the user is not 'seeing' the interrupts. This in addition to the fact that an interrupt cannot automatically be assumed to have come from the SCC or timer means significant overhead and rewriting of the //e and //c AppleTalk firmware is necessary. To recover a lot of the time lost in testing for where the interrupt came from and saving the environment, Columbia's firmware interrupt handler will switch to high speed operation (2.6 MHZ) as soon as possible after an interrupt occurs. Running faster should allow the AppleTalk firmware to get to the SCC before the SCC's FIFO overflows (approximately 104.167 microseconds after the initial SCC interrupt).

AppleTalk needs two major items of hardware to work with the 65816. An SCC chip and some hardware to provide 1/4 second interrupts. The SCC chip is Zilog's 8530 Serial Communications Controller chip running in SDLC (Synchronous Data Link Control) mode with FM0 encoding techniques. It outputs a beginning flag, data, 2 CRC (Cyclic Redundancy Check) bytes and a closing flag. Transmission frequency is 230.4K bits per second or 28,800 baud. A byte time is 34.722 microseconds or 4.340278 microseconds per bit.

Two timer functions are required for AppleTalk //e. One timer's function is to interrupt the 65816 every 1/4 (quarter) second for higher level protocol timing functions. This timing function will be executed by enabling a 3.75 hertz interrupt function of the Mega // chip. This will give the 65816 an interrupt every 1/3.75 seconds (0.266667 seconds). This will be used as the timing interrupt. The other timing function is to provide 400+xx microseconds delay for the inter-dialogue gap. This timing function will be executed in software.

It is assumed that the reader of this document is familiar with AppleTalk either on an Apple // product or on a Macintosh product.

Currently the firmware RAM used as a receive and write buffer will be mapped as follows.

<u>Memory Map of Columbia AppleTalk Firmware RAM</u>

```
             _ ______________________  <<---------------------------------
$E1xxxx       |                      |
              |                      |
              |                      |      Receive buffer (605 bytes)
              |                      |
              |                      |
$E1xxxx      _|______________________| <<---------------------------------
              |                      |
              |                      |
              |                      |      Write buffer (603 bytes)
              |                      |
              |                      |
$E1xxxx      _|______________________| <<---------------------------------
              |                      |
              |                      |      Work buffer (100 bytes)
              |                      |
$E1xxxx      _|______________________| <<---------------------------------
```

## Pointers, ID bytes, Entry points to AppleTalk

The following flags and pointers will be set up in slot 7 ROM area in Columbia starting at location $C700.

| Address | Purpose |
|---|---|
| $C705 | $38 Identifier byte #1 |
| $C707 | $18 Identifier byte #2 |
| $C70B | $01 Generic signature byte |
| $C70C | $9B Device signature byte<br>9 = Network or bus interface card/firmware<br>B = Apple Tech Support ID nybble |
| $C70D | $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 |

## Other Addresses Used by AppleTalk in Columbia

| Address | Purpose |
|---|---|
| $E0C038 | SCCADATA register |
| $E0C039 | SCCAREG register |
| $E0C03A | SCCBDATA register |
| $E0C03B | SCCBREG register |
| $E0C0xx | Enable 1/4 second timer interrupt |
| $E0C0xx | 1/4 second timer startus |
| $bb047F | User sets to $Cn ($C7 for Columbia) 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

## 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 // that AppleTalk uses is

disabled.

## Bootable AppleTalk General Information

1. AppleTalk in Columbia can be booted in one of 3 ways
    1. The MENU program options to start-up from internal slot 7 has been chosen.
    2. The student types in IN#7 or CALL 50965 from basic.
    3. The user types in $C715G from the monitor or JMP's or JSR's to $C715 from a program.

2. During booting the following occurs.
    1. A series of transfers between the AppleTalk firmware and main system RAM will occur. The higher level protocol necessary to request boot information from the master station is being moved from Columbia 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 with execution address of $200. This allows all memory from $800-$BFFF to be used for storing the main boot program loaded from the master station.
    2. When transfer is complete the AppleTalk boot code will jump to $200.
    3. The RAM code will establish communications with the master/teacher station and request the main boot code. This boot code could be PRODOS or PASCAL or whatever. Once the boot code is loaded, the RAM code will cause the boot code to begin execution.
    4. At this point in time the slave station is a fully operational system which will access files, at the master station, and a print station via AppleTalk assuming FAP and PAP have been loaded with the operating system. Initially the slaves will not be able to communicate between themselves, however it should not be long before some student writes the necessary code to do this. It can be done since the open architecture of AppleTalk does not prevent anyone from replacing our code with their code.

## Bootable AppleTalk Specific Information

Boot frames used for a normal boot sequence are as follows.

### Boot Request Frame

```
|------------------|
|  Destination adr |
|------------------|
|    Source adr    |
|------------------|
|     Lap type     |
|------------------|
| 0 0|Hop Cntl msb |
|------------------|
|lsb of Data Length|
|------------------|
|    Boot Type     |
|------------------|
|Block # Requested |
|------------------|
```

### Boot Information Response Frame

```
|------------------|
|  Destination adr |
|------------------|
|    Source adr    |
|------------------|
|     Lap type     |
|------------------|
| 0 0|Hop Cntl msb |
|------------------|
|lsb of Data Length|
|------------------|
|    Boot Type     |
|------------------|
|# Blks in bt Prog |
|------------------|
|...Place.Data.....|
|          Address |
|------------------|
|...Execution......|
|          Address |
|------------------|
```

Boot Response Frame

```
|------------------|
| Destination adr  |
|------------------|
|    Source adr    |
|------------------|
|     Lap type     |
|------------------|
| 0 0|Hop Cntl msb  |
|------------------|
|lsb of Data Length|
|------------------|
|    Boot Type     |
|------------------|
|   Block # Sent   |
|------------------|
|                  |
|      Block       |
.                  .
.       of         .
.                  .
|     Program      |
|                  |
|------------------|
```

## Frame Definitions

The boot request frame is used by the slave station to ask for boot information, to be sent all the boot blocks and to be sent specific boot blocks.

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

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

## Bytes Within Frames Definitions

Destination address for the Boot Request Frame is initially $FF, since a station coming on line doesn't know what the master's station # is.

Source address is the sending station's address #.

Lap Type is $08 for all boot transaction sequences.

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.

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.

Boot Type is defined as follows.

| | |
|---|---|
| 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 exactly 512 bytes.

Place data address is the address of where the slave station should start putting the main boot program at as it receives it from the master station.

Execution address is the address the boot program should jump to, to start up the main boot program.

## Columbia Boot Routine Memory Map

```
|-----------------|$00FFFF
.                 .
.                 .
.                 .
|-----------------|$000800
|                 |
|   Text Page 1   |  Block Byte Map
|                 |
|-----------------|$000400
| ROM             |
|...Boot.Code.....|$000300
|     Placed here |
|                 |
|-----------------|$000200
|                 |
|      Stack      |
|                 |
|-----------------|$000100
|                 |
|    Zero Page    |
|                 |
|-----------------|$000000
```

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

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

Locations $00-$1F and locations $56-$FF are available for the ROMs boot program to use as it is loading the boot program from the master station.

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

Slave boot screens.

Initial Screen

```
 ______________________________
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                        ----- |
|                        \***/ |
|                         \*/  |
|                         / \  |
|                        /   \ |
|                        ----- |
|______________________________|
```

After a station # (node #) is determined the following screen appears.
## is the node number in hexadecimal taken by AppleTalk.

Second Screen

```
 ______________________________
|I                             |
|##                            |
|                              |
|                              |
|                              |
|                              |
|                        ----- |
|                        \* */ |
|                         \*/  |
|                         /*\  |
|                        /   \ |
|                        ----- |
|______________________________|
```

After the boot information frame is received the following frame appears.

<u>Third Screen</u>

```
 ______________________________________
|I....................................|
|##                                   |
|                                     |
|.....................                |
|                                     |
|                                     |
|                              -----  |
|                              \  */  |
|                               \*/   |
|                               /*\   |
|                              / * \  |
|                              -----  |
|_____________________________________|
```

After timeout occurs or after block 1 (the last block since 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.

<u>Fourth Screen</u>

```
 ______________________________________
|I.      .                 .          |
|##                                   |
|                                     |
|                                     |
|                                     |
|                                     |
|                              -----  |
|                              \   /  |
|                               \*/   |
|                               /*\   |
|                              /** \  |
|                              -----  |
|_____________________________________|
```

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.

<u>Final Screen</u>

```
 ____________________________________
|I                                   |
|##                                  |
|                                    |
|                                    |
|                                    |
|                                    |
|                              -----  |
|                              \   /  |
|                               \ /   |
|                               /*\   |
|                              /***\  |
|                              -----  |
|____________________________________|
```

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. Slave station broadcasts Boot Request Frame with a boot type of 0 to get the Boot Info Frame. He broadcasts it every 1/4 second until the master station responds.
4. Master sends Boot Info Frame to slave.
5. Slave sends directed packet to master asking for all boot frames (Boot Type=1).
6. Master sends packets (blocks) sequentially '1' time only.
7. Slave receives frames and places them in sequential order in memory according to their block #.
8. Slave determines which blocks he missed.
9. Slave requests 1 at a time the block #'s he missed, waiting 150 msec between requests for missing frames.
10. Master sends requested blocks to slave.
11. Slave initializes the AppleTalk firmware.
12. Slave JMP's to Execution Address.
13. Program just loaded in takes control of the slave station.

e

s that no user RAM be occupied with any of the ATLAP code.
n such a way as to assure an identical interface between
mbia. This interface will allow the user to write
 protocols (such as a new DDP) and still be able to use our
neric LAP protocol interface will allow us at any time to
P software/hardware without requiring the applications
 any changes to their programs. The firmware entry points
ation in the $Cn00 ($C700 in Columbia) space which is
he //e and //c versions.

e LAP as follows;

ie in that is requires only 1 entry point into the $Cn00
t future maintainability is simple. We only need to make
( entry point is maintained.

```
.ST   ;Y = hi byte of parameter list address
.ST   ;X = lo byte of parameter list address
      ;A = the slot # of the AppleTalk interface+$C0
          ($C7 in Columbia)
.K    ;Call the interface (in ROM in //c and in Columbia
       and in RAM section of peripheral card in //e)
NE    ;<>0 then an error occurred
```

l always be clear upon exit from the AppleTalk

AppleTalk Generic PARAMLST is defined as follows

```
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
```

PARAMLST's Defined for Each Call

INIT Call - 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 the
                               user set up Columbia as a normal node
                               or as a server node.
                       4. returns AppleTalk station address
```

READREST Call - Command Number 2

```
    DFB $2          ;Command number for READREST call
    DW BUFFADDR     ;Address in user's program for rest of data packet
                     to be put
                       1. Address of read buffer (buffer for packet
                          to be transferred to)
    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 to)
         DS 2,0           ;Number of bytes read in during READREST call.
```

<u>WRITE Call - Command Number 3</u>

```
         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 table

                     WRITETBL EQU *     ;Generic form
                        DW NUMDATABYTES ;Number of bytes to read
                        DW DATABUFFER   ;Pointer to data buffer
                        DW NUMDATABYTS2 ;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 Call - 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 does 2 things.  It sets interrupt
                          masks and returns interrupt status to the user.
                          If STATUS is called with a parameter byte of
                          - then the call is setting 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 = 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 pkt or/and timer event
                                     occurred
                          B6 = 0/1   1/4 sec timer went off
                          B5-B4      RESERVED
                          B3-B0      1 bit set for each pkt in buffer (1
                                     packet maximum in Columbia)
```

## READPROT Call - Command Number 5

```
     DFB $5              ;Command number for READPROT call
     DW BUFFADDR         ;Address in users's program for part of data
                          packet to be put
                            1. Address of read buffer (buffer for packet
                               to be transferred to)
     DS 2,0              ;Number of bytes
                            1. Number of bytes to read
```

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

NOTE

For all calls carry will return set if an error occurred and the accumulator will contain the error code.

For a STATUS call carry will return set (indicating the user was in error assuming that the AppleTalk was the interrupting device) if AppleTalk was not the interrupting device. Carry will return clear if AppleTalk was the interrupting source (indicating the user was correct in assuming the AppleTalk was the interrupting source.

## Error Codes by Call Number

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

INIT call errors
- 4 = could not get unique AppleTalk address for station or in the //c version could not talk to the AppleTalk //c protocol converter box.

READPROT call errors
- 1 = no packets in buffer to read
- 2 = multipurpose buffer overflowed (not possible in Columbia)
- 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 Columbia)

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 interrupt mask call errors
- None possible

## Brief Description of Each Call

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

NOTE: The user must call STATUS with an interrupt mask to enable AppleTalk interrupts to be passed to the user.

```
The INIT call returns:     C = 0 if no error
                           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 occurred
                              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 done 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 occurred
                              C = 1 if an error occurred
                              A = error code
                      X / Y / V = scrambled
```

WRITE: Called by appropriate protocol level to move data from protocols buffer and send a datagram on AppleTalk. WRITE passes a pointer to a table of pointers and byte counts that included sequentially, comprise 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 out by the Appletalk firmware. Therefore the user does not need to know his station (node) number to transmit a packet. The user must however provide space for the source address to go when he is defining a packet.

```
The WRITE call returns:    C = 0 if no errors
                           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 or not. If C=0 it was, C=1 if it was not. Also returns whether it was a 1/4 second timer interrupt or a packet ready interrupt if AppleTalk was the interrupting device. If an AppleTalk source was not the interrupting device the accumulator register returns with a $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 Columbia's AppleTalk ability to be used in a polling mode fashion 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 to work.

```
The STATUS call returns: C = 0 if AppleTalk was interrupting
                               device (Clears interrupt)
                         C = 1 if AppleTalk was not the
                               interrupting device
                         A = error code
                     X / Y / V = scrambled
```

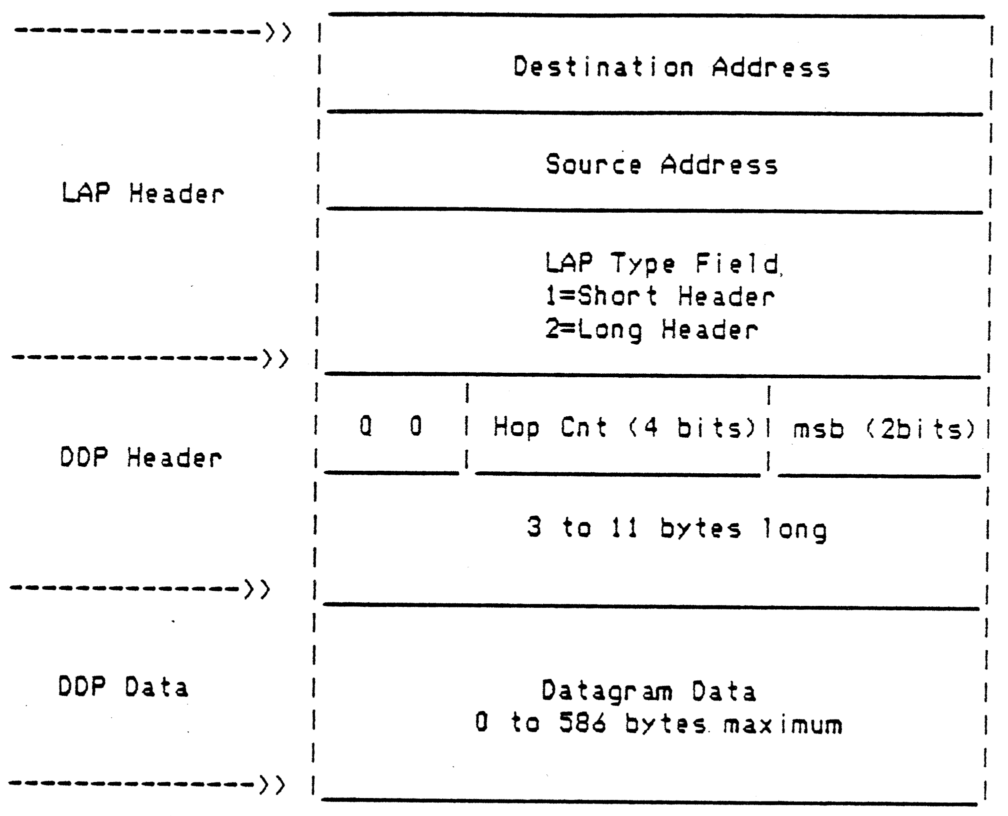
Apple // AppleTalk Interface General Diagram

```
|----------------------Apple-------------------|------Peripheral Card-----|
|                      //e                     |                          |
  _________________________________            ____________________________
|            |                    |            |              |           |
|            |                    |____________|    ATLAP     |           |
|    O.S.    |    AppleTalk       | Peripheral |    65C02     |    SCC    |ATLK
|            |    Protocol        |   Card     |    2K RAM    |    I/F    |----
|            |                    |    Bus     |  4K(8K)ROM   |           |
|____________|____________________|____________|______________|___________|
                                               |
                            ATLAP layer as -->>|
                            seen by DDP layer

|---------------Apple------------|-------Protocol Converter Box----------|
|               //c              |                                      |
  ________________________|___________               _________________________
|            |           |       |       |                 |              |           |
|            |ApTalk     | ROM   | //c   |___         ___|    ATLAP     |           |
|    O.S.    |           | PRE-  | Prot  | I |       | I |    65C02     |    SCC    |ATLK
|            | Prot      | LAP   | Conv  | W |---| W |    2K RAM    |    I/F    |----
|            |           |       |       | M |       | M |  4K(8K)ROM   |           |
|____________|___________|_______|_______|___|       |___|______________|___________|
                         |                                    |
ATLAP layer as -->>|                                          |
seen by DDP layer                                             |
                                                            CBUS

|--------------------------------Columbia--------------------------------|
|                                                                        |
  ________________________________  __________________________________________
|            |                    |                              |           |
|            |                    |                              |           |
|    O.S.    |    AppleTalk       |         ROM ATLAP            |    SCC    |ATLK
|            |    Protocol        |      with RAM buffers        |    I/F    |----
|            |                    |                              |           |
|____________|____________________|______________________________|___________|
                                  |
              ATLAP layer as -->>|
              seen by DDP layer
```

## Receive Buffer Columbia

During an interrupt to the 65816, the firmware interrupt handler will determine if it is an AppleTalk related interrupt. If it is it will call AppleTalk firmware to handle the interrupt, read data into the receive buffer, and call the user if required to. When the user is interrupted he will call the routine called STATUS to determine what type of AppleTalk interrupt 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 Data Structure a Packet



## Automatic Packet Rejection by the AppleTalk Firmware

The firmware will automatically reject an incoming packet under the following conditions:

1. Any SCC error occurs.
   Including: receiver overrun, CRC, missing clocks
2. More than 603 bytes are in incoming packet
3. The number of bytes-3 received do not equal the length byte parameters in byte positions 3 and 4 in the packet just received.
4. No characters received within 1 character time. (approximately 34.722 microseconds.
5. A WRITE is in progress.

In every case the user is not interrupted if any of the above conditions occur. The firmware simply resets its pointers and waits for more packets to be sent to it.

## Interrupting the User

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

The interruptability of the user is totally dependant on the interrupt mask sent to the AppleTalk firmware during the last STATUS call used to set the interrupt mask. It can be set to allow timer interrupts or/and packet ready interrupts in any combination.

It is possible, although not using our higher level drivers, to use AppleTalk in non-user interrupt mode by polling the AppleTalk firmware. This is done by periodically doing a STATUS call ignoring the carry bit and decoding the status byte. If bit 7 is set an AppleTalk event occurred. Bit 6 is set if the 1/4 sec timer went off. Bits 0 will be set to indicate a packet was received since the last READREST call. Using this data the user can call READPROT and READREST to extract the packet data from AppleTalk's firmware RAM buffer.

<u>NOTE:</u> For AppleTalk on Columbia to work interrupts must be enabled, whether the user wants to be interrupted or not. If the user does not want to be interrupted the firmware will trap, decode, and act on all AppleTalk interrupt sources transparent to the user.

## Resetting the AppleTalk Firmware / Hardware

AppleTalk Firmware and Hardware can be reset in 3 ways.

1. Pressing 'CONTROL-RESET'.
2. Pressing 'CONTROL-open apple-RESET
3. Powering up the system

## lapENQ, lapACK, lapRTS, lapCTS

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

## Miscellaneous Comments

The AppleTalk firmware will be made recognizable with appropriate ID bytes for PRODOS and PASCAL. Although AppleTalk // is using the generic PASCAL 1.1 firmware entry points, 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 those operating systems to access AppleTalk.

The AppleTalk PRODOS driver(s) will reside in the main language card bank 2 at $D400-$DFFF. The AppleTalk driver for PASCAL will reside on the heap.

## Printer Hooks Via the AppleTalk Firmware

There is no room in the AppleTalk firmware to provide all the protocol and routines necessary to output to a print server. However by providing proper hooks in the AppleTalk interface firmware we can redirect the users (or application programs) printer outputs to a printer driver located in main memory in the Apple //. The scheme we have chosen should allow BASIC and PRODOS application programs to access the AppleTalk interface firmware as if it were a normal printer 'card'. That means that 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 means that the user wants to initialize the printer driver interface if one is loaded into main memory. To determine whether a driver is available or not we must perform the following step.

> Test the 1st screen hole $47F to verify that it is $C7 ($C7 is the flag indicating 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 goes to the driver this way;

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.
4. Loads the printer driver bank address from screen hole $6FF and pushes it on the stack.
5. Does an RTS which goes to the driver if shadowing is on. Does an RTL which goes to the driver if shadowing is off.

The AppleTalk interface firmware passes information in the following form 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 for the driver itself will be to place $Cn at screen hole location $47F and its execution address-1 into screen holes $77F (low byte) and $77F (high byte) and $6FF (bank byte).