```
PAGE -   0
Current memory available:     512
0000|                          .title  "PCcard - firmware for IBM-PC AppleTalk"
0000|                     ;*****************************************************************************
0000|                     ;*  This is the master file for assembling the AppleTalk PC-card ROM.
0000|                     ;*
0000|                     ;*  Modification history:
0000|                     ;*
0000|                     ;*  08/10/88 RRH
0000|                     ;*      ECO, 34x-0007-B14
0000|                     ;*          fixed initialization code for SCC.
0000|                     ;*
0000|                     ;*  05/29/87 RRH
0000|                     ;*      ECO, 34x-0007-B12, B13
0000|                     ;*          fixed checksum bug in pcDDP.
0000|                     ;*
0000|                     ;*  01/09/87 RRH
0000|                     ;*      ECO, 34x-0007-B11
0000|                     ;*          changed freeDMA to use SCCctrl for fDMA clear.
0000|                     ;*
0000|                     ;*  12/01/86 RRH
0000|                     ;*      ECO, 34x-0007-B10
0000|                     ;*          changed SCCintr code for "lost" interrupts.
0000|                     ;*          added CMDhst for logging host commands.
0000|                     ;*
0000|                     ;*  11/18/86 RRH
0000|                     ;*      ECO, 34x-0007-B9
0000|                     ;*          adjusted txLAP timings;  16 abort bits, 200 µsec
0000|                     ;*          RTS-DATA Broadcast IFG.
0000|                     ;*
0000|                     ;*  11/13/86 RRH
0000|                     ;*      ECO, 342-0007-B8
0000|                     ;*          fixed length calculation in rxLAP, so rxDDP works!
0000|                     ;*
0000|                     ;*  10/31/86 RRH
0000|                     ;*      ECO, 342-0007-B6/7
0000|                     ;*          added pcDBG,
0000|                     ;*          added CheckSumming call ($25) for DDP
0000|                     ;*
0000|                     ;*  8/21/86  RRH
0000|                     ;*      ECO, 342-0007-B5
0000|                     ;*          removed creation of TTTBL on XO TReq;
0000|                     ;*
0000|                     ;*  5/27/86  RRH
0000|
0000|                     ;*      ECO, 342-0007-B4
0000|                     ;*          fixed nRXREQ timer bug (not setting ttTmr)
0000|                     ;*          fixed bldHdr bug (skipping CKS, sNode)
0000|                     ;*
0000|                     ;*  3/13/86  RRH
0000|                     ;*      ECO, 342-0007-B3
0000|                     ;*          fixed filtering;
0000|                     ;*          deleted auto-matic TID generation
0000|                     ;*  3/12/86  RRH
0000|                     ;*      ECO, 342-0007-B2; changes:
0000|                     ;*          added filtering (SysFlgs)
>>>>PAGE -   1             FILE:   PCcard - firmware for IBM-PC AppleTalk


0000|                     ;*
0000|                     ;*  3/1/86   RRH
0000|                     ;*      ECO, 342-0007-B1; changes:
0000|                     ;*          changed to create TTTBL entry on rcv of XO TReq;
0000|                     ;*          use this entry on SENDRSP.
0000|                     ;*  12/8/85  RRH
0000|                     ;*      ECO, 342-0007-B;  changes:
0000|                     ;*          use chanB DCD for pcDMA.
0000|                     ;*          change retry DDP to use ATP timing logic.
0000|                     ;*          fixed SENDNBP TmOutErr leaving nbTTP set
0000|                     ;*
0000|                     ;*   9/10/85 RRH
0000|                     ;*      ECO, 342-0007-A;  changes:
0000|                     ;*          reset ttTmr when STS reply recv'd for SendResp.
0000|                     ;*          pass XO flag on TResp packets.
0000|                     ;*          prevent fake DMAinfo on TRel send.
0000|                     ;*          modified DDP/ATP as per changes (4/26/85).
0000|                     ;*      ECO, 342-0007-A;  additions:
0000|                     ;*          RTMP handling.
0000|                     ;*          DDP transmit retry mechanism
0000|                     ;*
0000|                     ;*   8/17/85 RRH
0000|                     ;*      initial release of ROM, 342-0007.
0000|                     ;*
0000|                     ;*  Copyright Notice:
0000|                     ;*      (C) Apple Computer, Inc., 1985, 1986, 1987, 1988
0000|
0000|                     ;*      All rights reserved.
0000|                     ;*  Ron Hochsprung, 8/10/88
```

```
0000|                   ;*****************************************************************************
0000|
0000|                       .absolute
0000| 0001              fROM    .equ    1       ; 0 -> ROMulator, 1 -> real ROM
0000|
0000|                       .include    PCDEFS      ; Symbolic equates for PCcard
0000|                   ;***    PCDefs.text, definitions file for IBM-PC AppleBus card
0000|
0000|                   ;*****************************************************************************
0000|                   ;** basic hardware base addresses
0000|                   ;*****************************************************************************
0000| 4000              sccCtrl .equ    04000           ; control address (B-port)
0000| 4001              sccData .equ    04001           ; data       "       "
0000| 4002              sccCtrlA  .equ  04002           ; control    "    (A-port)
0000| 4003              sccDataA  .equ  04003           ; data       "       "
0000| 4002              DBGctrl .equ    04002           ; DeBuG port (A)
0000| 4003              DBGdata .equ    04003           ;   "
0000| 8000              DMAreg  .equ    08000           ; DMA (IBM bus) address space
0000| C000              STAtreg .equ    0C000           ; Status reg -> IBM
0000| E000              ROMbase .equ    0E000           ; starting address for ROM
0000|
0000|                   ;*****************************************************************************
0000|                   ;*  STATreg values.  These values are used to indicate to the PC what
0000|                   ;* our code expects the DMA channel to be setup with.  When nothing is
0000|                   ;* expected, the STSidle value is used.  Note that values are in upper
0000|                   ;* nybble, even tho they show up in low one on PC;  that's the way the dumb
>>>>PAGE -   2            FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000|                   ;* hardware works.....
0000|
0000|                   ;*****************************************************************************
0000| 0080              DMAidle .equ    080         ; idle status
0000| 0090              STSrdy  .equ    090         ; status is available (REQdone)
0000| 00A0              REQinfo .equ    0A0         ; Request Info (PC) -> card
0000| 00B0              REQdata .equ    0B0         ; Request Data (PC) <-> card
0000| 00C0              DMAinfo .equ    0C0         ; DMA req -> PC
0000| 00D0              DMAdata .equ    0D0         ; DMA data <-> PC
0000|
0000|                   ;*****************************************************************************
0000|                   ;*  SCC related addresses
0000|                   ;*****************************************************************************
0000|                   ;*  ReadReg 0 defs
0000| 0001              RCA .equ    00000001t       ; Rx Char Avail
0000| 0004              TBE .equ    00000100t       ; Tx Bfr Empty
0000| 0008              DCD .equ    00001000t       ; Data Carrier Detect
0000| 0010              HUNT    .equ    00010000t       ; Hunt mode (0-> in packet)
0000| 0020              CTS .equ    00100000t       ; ClearToSend
0000| 0040              EOM .equ    01000000t       ; EndOfMessage (Tx UnderRun)
0000| 0080              ABORT   .equ    10000000t       ; Abort sequence detected
0000|
0000|                   ;** ReadReg 1 defs
0000| 0020              OVR .equ    00100000t       ; Rx OverRun
0000| 0040              CRC .equ    01000000t       ; Rx CRC error
0000| 0080              EOF .equ    10000000t       ; Rx EndOfFrame
0000|
0000|                   ;** ReadReg 10 defs
0000| 00C0              MCmask  .equ    11000000t       ; missing clock mask (1 / 2 clocks)
0000|
0000|                   ;*****************************************************************************
0000|                   ;*  Link Access Protocol (LAP) definitions.  (offsets)
0000|                   ;*****************************************************************************
0000| 0000              laDst   .equ    0           ; offsets of LAP info
0000| 0001              laSrc   .equ    1
0000| 0002              laType  .equ    2           ; encapsulated type
0000| 0003              laData  .equ    3           ; offset of start of user data
0000| 0001              laDDP       .equ    001         ; normal (short) DDP
0000| 0002              laDDPX      .equ    002         ; extended (long, internet) DDP
0000| 0081              laENQ       .equ    081         ; ENQUIRY, auto-node assignment
0000| 0082              laACK       .equ    082         ; ACKNOWLEDGE, response to ENQ
0000| 0083              laNAK       .equ    083         ; Neg-ACK (maybe, some day)
0000| 0084              laRTS       .equ    084         ; RequestToSend
0000| 0085              laCTS       .equ    085         ; ClearToSend
0000|
0000|                   ;*****************************************************************************
0000|                   ;*  Directed Datagram Protocol (DDP) definitions.
0000|                   ;* offsets are from DDP header;  must be further offset by laData.
0000|                   ;* "ddXXX" names are for short DDP, "xdXXX" for DDPX.
0000|                   ;*****************************************************************************
0000|
0000| 0000              ddLng   .equ    0           ; length field (12 bits)
0000| 0002              ddDskt  .equ    2           ; dest socket number
0000| 0003              ddSskt  .equ    3           ; source "      "
0000| 0004              ddType  .equ    4           ; encapuslated type
>>>>PAGE -   3            FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000| 0001              ddRTMP      .equ    001         ; Routing Table Maintenance
```

```
0000| 0002              ddNBP       .equ    002         ; Name Binding
0000| 0003              ddATP       .equ    003         ; AppleTalk Transaction
0000| 0005              ddRTMP1     .equ    005         ; RTMP+             ;-A
0000| 0005              ddData  .equ    5           ; user data
0000|
0000| 0000              xdHopCt .equ    0           ; hop count (4 bits)
0000| 0000              xdLng   .equ    0           ; length field (12 bits)
0000| 0002              xdCKS   .equ    2           ; checksum (2 bytes)
0000| 0004              xdDnet  .equ    4           ; dest network number (2 bytes)
0000| 0006              xdSnet  .equ    6           ; source  "       "
0000| 0008              xdDnode .equ    8           ; dest node address
0000| 0009              xdSnode .equ    9           ; source "      "
0000| 000A              xdDskt  .equ   10.          ; dest socket number
0000| 000B              xdSskt  .equ   11.          ; source  "     "
0000| 000C              xdType  .equ   12.          ; encapsulated type
0000| 000D              xdData  .equ   13.          ; user data
0000|
0000|                   ;***************************************************************************
0000|                   ;*  Routing Table Maintenance Protocol (RTMP) definitions.
0000|                   ;***************************************************************************
0000| 0000              rtmNet  .equ    0.          ; offset of net#    ;-A
0000|
0000| 0001              rtmSKT  .equ    1.          ; Socket # for RTMP ;-A
0000| 0001              rtmCMD1 .equ    1.          ; command for RTMP+ ;-A
0000|
0000|                   ;***************************************************************************
0000|                   ;*  Name Binding Protocol (NBP) definitions.
0000|                   ;***************************************************************************
0000| 0002              nbNIS   .equ    2.          ; Socket # for all DDP datagrams
0000|
0000|                   ;***************************************************************************
0000|                   ;*  AppleTalk Transaction Protocol (ATP) definitions.
0000|                   ;* offsets are from ATP header;  must be offset by ddData/xdData.
0000|                   ;***************************************************************************
0000| 0000              atCCI   .equ    0           ; Command/Control Info
0000| 00C0              atCmsk      .equ    11000000t       ; mask for Control bits
0000| 0040              atREQ       .equ    01000000t       ; REQUEST code
0000| 0080              atRSP       .equ    10000000t       ; RESPONSE code
0000| 00C0              atREL       .equ    11000000t       ; RELEASE code
0000| 0020              atXO        .equ    00100000t       ; eXactly Once
0000| 0010              atEOM       .equ    00010000t       ; End Of Message
0000| 0008              atSTS       .equ    00001000t       ; STatuS request
0000| 0001              atBtMap .equ    1           ; Bit map (TReq)
0000| 0001              atSqNbr .equ    atBtMap         ; Sequence Number (TResp)
0000|
0000| 0002              atTID   .equ    2           ; Transaction ID
0000| 0004              atUser  .equ    4           ; 4 bytes of "user" data
0000| 0008              atData  .equ    8           ; data starts here
0000|
0000|                   ;***************************************************************************
0000|                   ;*  Task table defn's
0000|                   ;* These definitions reflect the organization of the task table.  There is
0000|                   ;* on entry for each of the system's tasks.
0000|                   ;***************************************************************************
>>>>PAGE -   4            FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000|
0000| 0000              tFlags  .equ    0       ; Task flags
0000| 0080              tREADY      .equ    10000000t   ;   "  " running (at least, ready to run)
0000| 0001              tWaits  .equ    1       ; what task is waiting for
0000| 0001              sSEMA       .equ    00000001t   ; task is waiting on semaphore (all)
0000| 0002              sNEWREQ     .equ    00000010t   ;   "  "     "    for a Host req (ATP)
0000| 0004              sTXDONE     .equ    00000100t   ;   "  "     "    for XMTdone (CMD/ATP)
0000| 0040              sONESEC     .equ    01000000t   ;   "  "     "     "  one-second intr (ATP)
0000|
0000| 0002              tNext   .equ    2       ; Next Task on a queue (e.g., semaphore)
0000| 0003              tSP .equ    3       ; Stack pointer for this task
0000| 0004              tBufN   .equ    4       ; buffer for this Task (may be 1st of queue)
0000| 0005              tRslt   .equ    5       ; result of request
0000| 0007              tRID    .equ    7       ; Request ID of this Task
0000|
0000| 0008              tSize   .equ    8       ; length of each entry.
0000|
0000| 0000              sCnt    .equ    0       ; Semaphore counter
0000| 0001              sQhd    .equ    1       ; Queue header for semaphore
0000| 0002              sQtl    .equ    2       ;   "   tail    "     "
0000| 0003              sOwn    .equ    3       ; Semaphore owner (0FF if none)
0000|
0000| 0000              t0  .equ    000         ; a temp
0000| 0001              t1  .equ    001         ;  another..
0000| 0002              t2  .equ    002
0000| 0003              t3  .equ    003
0000| 0004              t4  .equ    004
0000| 0005              t5  .equ    005
0000| 0006              t6  .equ    006
0000| 0007              t7  .equ    007
0000| 0008              t8  .equ    008
```

```
0000| 0009              t9  .equ    009
0000|
0000| 000B              aBridge .equ    00B     ; a Bridge on our Network
0000| 000C              ThisNet .equ    00C     ; our network # (determined from RMTP)
0000|
0000| 0010              pcFlags .equ    010     ; flags relating to PC actions
0000| 0020              fREQ        .equ    CTS     ; PC is requesting service
0000| 0008              fDMA        .equ    DCD     ; PC says DMA is now setup
0000| 0011              pcIdle  .equ    011     ; idle state status (either DMAidle/STSrdy)
0000| 0012              pcREQ   .equ    012     ; REQcode from PC
0000| 0013              myFlags .equ    013     ; some flags for our usage
0000| 0080              fMyNode     .equ    10000000t   ; myNode is valid
0000|
0000|                   ;** warning codes (these are not "fatal")
0000| 0001              SKTerr      .equ    001     ; Socket/protocol error
0000| 0002              ABTdone     .equ    002     ; ABT finished
0000|
0000|                   ;** error codes
0000| FFFF              REQerr      .equ    -001    ; REQuest is illegal
0000| FFFE              NoFreeErr   .equ    -002    ; no Free buffer
0000| FFFD              TmOutErr    .equ    -003    ; REQ/RSP timed out
0000|
0000| FFFC              NoTTErr     .equ    -004    ; no TTtbl entry for this RID
0000| FFEF              DeferErr    .equ    -011    ; excess Defers (txPacket)
>>>>PAGE -   5            FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000| FFEE              ColsnErr    .equ    -012    ;   "    Collisions  "
0000|
0000| 0014              rtP .equ    014     ; RCVtask's ptr (to packet)
0000| 0018              dP  .equ    018     ; DMA pointer
0000| 001A              dC  .equ    01A     ;  "  counter
0000| 001C              TMRsecs .equ    01C     ; seconds value (16-bits)       ;-A
0000| 001E              RTMPtmr .equ    01E     ; timer for RTMP invalidation       ;-A
0000| 001F              ATPsecs .equ    01F     ; last TMRsecs that ATP was updated
0000|
0000|                   ;*****************************************************************************
0000|                   ;*  stuff used by XMTtask
0000|                   ;*****************************************************************************
0000| 0020              XMTsema .equ    020     ; the semaphore for XMTtask
0000| 0024              xmitQ   .equ    024     ; queue for XMTtask
0000| 0024              xmitQhd .equ    xmitQ
0000| 0025              xmitQtl .equ    xmitQ+1
0000| 0026              ctP .equ    026     ; working buffer ptr for XMTtask
0000|
0000| 002F              CMDhstP .equ    02F     ; index into CMDhst
0000| 1F80              CMDhst  .equ    01F80       ; Command History table
0000|
0000| 0030              SysFlgs .equ    030     ; System Flags (protocol overrides, etc.)
0000| 0001              fNLAP   .equ    00000001t   ; 1-> No LAP pkts (no receives!!)
0000| 0002              fALAP   .equ    00000010t   ; 1-> All LAP packets (no filtering)
0000| 0004              fNDDP   .equ    00000100t   ; 1-> No DDP processing (all LAP)
0000|
0000| 0008              fADDP   .equ    00001000t   ; 1-> All DDP (no skt filtering)
0000| 0010              fNATP   .equ    00010000t   ; 1-> No ATP processing
0000|
0000|                   ;*****************************************************************************
0000|                   ;*  Task control Tables for our tasks.
0000|                   ;*****************************************************************************
0000| 0040              CMDtcb  .equ    040         ; Host I/F task
0000| 0048              XMTtcb  .equ    048         ; Transmit task
0000| 0050              RCVtcb  .equ    050         ; Receive task
0000| 0058              ATPtcb  .equ    058         ; ATP task
0000|
0000| 0040              CMDFlgs .equ    CMDtcb+tFlags       ; special symbols for shorthands
0000| 0044              CMDBufN .equ    CMDtcb+tBufN
0000| 0047              CMDRID  .equ    CMDtcb+tRID
0000| 0045              CMDRslt .equ    CMDtcb+tRslt
0000| 004C              XMTBufN .equ    XMTtcb+tBufN
0000| 0054              RCVBufN .equ    RCVtcb+tBufN
0000| 0058              ATPFlgs .equ    ATPtcb+tFlags
0000| 005C              ATPBufN .equ    ATPtcb+tBufN
0000|
0000|                   ;** main task control params here
0000| 0060              theTask .equ    060     ; current task, 0 if none active
0000| 0061              TaskQ   .equ    061     ; task queue header
0000| 0061              TaskQhd .equ    TaskQ
0000| 0062              TaskQtl .equ    TaskQ+1
0000| 0063              theParm .equ    063     ; current Parameter Ptr
0000|
0000|                   ;** working packets for Tx/Rx
0000| 0068              RTSfrm  .equ    068     ; 3-byte area for Tx LAP control frames
>>>>PAGE -   6            FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000| 0069              myNode  .equ    RTSfrm+laSrc    ; our node number
0000| 006C              CTSfrm  .equ    06C     ; 3-bytes for RTS/ENQ responses
0000|
```

```
0000|
0000|                  ;*************************************************************************
0000|                  ;*  The big 4 semaphores, about which the flow of a task's execution
0000|                  ;* revolves.
0000|                  ;*************************************************************************
0000| 0070             REQsema .equ    070     ; semaphore for incoming Requests
0000| 0074             DMAlock .equ    074     ;     "       "  DMA access (owns Status Reg)
0000| 0078             XMTlock .equ    078     ;     "       "  Transmit
0000| 007C             ATPlock .equ    07C     ;     "       "  TT modification
0000|
0000|                  ;*************************************************************************
0000|                  ;*  Free frame storage control.  We have room for 11 frame buffers (each
0000|                  ;* of length 640 bytes).  The available buffers are kept track of in freeQ.
0000|                  ;* The "next" field for each buffer are contained within the same area, not
0000|                  ;* in the buffers themselves.  Note that the buffers actually start in page
0000|                  ;* 0400.
0000|                  ;*************************************************************************
0000| 0080             freeQ   .equ    080     ; start of queue control area
0000| 0080             freeQhd .equ    freeQ
0000| 0081             freeQtl .equ    freeQ+1
0000| 0268             HDROFS  .equ    0268        ; offset of header in buffer
0000|
0000|
0000| 008E             STSQhd  .equ    08E     ; queue of result status's (STStbl = 0100)
0000| 008F             STSQtl  .equ    STSQhd+1
0000|
0000|                  ;*************************************************************************
0000|                  ;*  Tx variables.  These are used by the txPacket routines, whose access
0000|                  ;* is controlled via the XMTlock.  Only one task may be trying to transmit
0000|                  ;* a "packet" at one time.  Note, however, that txFrame may be called by
0000|                  ;* others (rxIntr).
0000|                  ;*************************************************************************
0000| 0090             txPh    .equ    090         ; header, pointer
0000| 0092             txCh    .equ    092         ;    "   , count
0000| 0094             txPd    .equ    094         ; data, pointer
0000| 0096             txCd    .equ    096         ;  "  , count
0000|
0000|                  ;*************************************************************************
0000|                  ;*  Rx variables.  rxP/rxC represent current free buffer info.  rP/rC
0000|                  ;* are working copies which the rxIntr routine uses.  In case of an error, or
0000|                  ;* when a new frame buffer is allocated, rP/rC are reset to values of rxP/rxC.
0000|                  ;*  If rxP(rP) is zero, no buffer is available.  Note that since all
0000|                  ;* buffers are allocated in non-zero page, checking rxP+1 is sufficient.
0000|
0000|                  ;*************************************************************************
0000| 00A0             rP  .equ    0A0         ; working copies of rxP, rxC
0000| 00A2             rC  .equ    0A2
0000| 00A4             rxP .equ    0A4         ; Rx pointer
0000| 0260             rxCsz   .equ    608.            ; max packet size (plus slop)
0000| 00A6             rxBufN  .equ    0A6         ; rxIntr's buffer number
0000|
0000| 00A8             RCVsema .equ    0A8         ; RCVtask's semaphore
>>>>PAGE -   7           FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000| 00AC             recvQ   .equ    0AC         ;     "     queue
0000| 00AC             recvQhd .equ    recvQ
0000| 00AD             recvQtl .equ    recvQ+1
0000|
0000| 00AE             fType   .equ    0AE         ; <>0 iff, a Frame was received
0000| 00AF             laState .equ    0AF         ; current LAP state
0000| 0000             laIdle      .equ    0           ; nothing happening
0000| 0001             laDATwt     .equ    1           ; waiting for Data (RTS seen, CTS sent)
0000| FFFF             laCTSwt     .equ    -1          ;   "      "  CTS (RTS sent)
0000|
0000|                  ;*************************************************************************
0000|                  ;*  global counters used for errors, statistics, etc.
0000|                  ;*************************************************************************
0000| 00B0             Counts  .equ    0B0         ; origin of counters
0000| 00B0             INTct   .equ    Counts          ; RCA interrupts
0000| 00B2             RXPct   .equ    INTct+2         ; valid Packets (data) Rx'd
0000| 00B4             TXPct   .equ    RXPct+2         ; Packets correctly(?) sent
0000|
0000|                  ;** errors detected by Rx code
0000|
0000| 00B6             RTSct   .equ    TXPct+2         ; RTS/ENQ
0000| 00B8             CTSct   .equ    RTSct+2         ; CTS/ACK
0000| 00BA             SKPct   .equ    CTSct+2         ; SKiPped packets
0000| 00BC             UNDct   .equ    SKPct+2         ; UNderRuns
0000| 00BE             OVRct   .equ    UNDct+2         ; OVerRuns
0000| 00C0             CRCct   .equ    OVRct+2         ; CRC errors
0000| 00C2             BADct   .equ    CRCct+2         ; Bad packets
0000| 00C4             LNGct   .equ    BADct+2         ; bad Length packets
0000| 00C6             DUPct   .equ    LNGct+2         ; Duplicate Node# detects
0000|
0000|                  ;*************************************************************************
0000|                  ;*  Stuff needed by SNDNBP.
0000|                  ;*************************************************************************
```

```
0000| 00CC            nbTTP   .equ    0CC         ; ptr to our TTP value      ;-B
0000| 00CD            nbTmOut .equ    nbTTP+1         ; temp save for timer       ;-B
0000| 00CE            nbRtrys .equ    nbTmOut+1       ;   "   "    "  retry count     ;-B
0000|
0000|                ;*******************************************************************************
0000|                ;*  Tx areas.  Note that xP/xC are arguments to txFrame routines.
0000|                ;* As such, they may change as the result of a received packet (to send the
0000|                ;* CTS);  hence, the txPacket routines must assure that they are reset before
0000|                ;* sending the RTS and/or DATA frames.
0000|                ;*******************************************************************************
0000|
0000| 00D0            xP  .equ    0D0         ; Tx pointer
0000| 00D2            xC  .equ    0D2         ;  " counter
0000| 00D4            xChist  .equ    0D4         ; Collision history
0000| 00D5            xCtrys  .equ    0D5         ;     "     attempts
0000| 00D6            xDhist  .equ    0D6         ; Defer history
0000| 00D7            xDtrys  .equ    0D7         ;   "   attempts
0000| 00D8            xGmask  .equ    0D8         ; Global mask
0000| 00D9            xLmask  .equ    0D9         ; Local mask
0000| 00DB            xfBcast .equ    0DB         ; <>0 if broadcast packet
0000| 00DD            xDelay  .equ    0DD         ; current delay value (100 usec units)
0000| 00DE            xSeed   .equ    0DE         ; current random value
>>>>PAGE -   8          FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000|
0000|                ;*******************************************************************************
0000|                ;*  most of 00E0 line is used for ATP related stuff..
0000|                ;*******************************************************************************
0000| 00E0            atP .equ    0E0         ; ptr to ATP buffer
0000| 00E2            atPh    .equ    0E2         ;  "   "  "    "  (header area)
0000| 00E4            atC .equ    0E4         ; count of data buffer
0000|
0000| 00E8            lstTTP  .equ    0E8         ; last allocated TT entry
0000| 00E9            lstATP  .equ    0E9         ; last modified entry (non-ATPtask)
0000|
0000| 00EC            atTXQ   .equ    0EC         ; TX Queue
0000| 00EC            atTXQhd .equ    atTXQ
0000| 00ED            atTXQtl .equ    atTXQhd+1
0000| 00EE            xmtATP  .equ    0EE         ; TT being XMT'd
0000|
0000|                ;*******************************************************************************
0000|                ;*  all of 00F0 line is used for DBG related stuff..
0000|
0000|                ;*******************************************************************************
0000| 00FF            NMIflg  .equ    0FF         ; Flag for NMI debounce
0000| 00FC            PCreg   .equ    0FC         ; saved PC
0000| 00FB            Preg    .equ    0FB         ; processor status
0000| 0080            Nbit        .equ    10000000T   ; Negative bit
0000| 0040            Vbit        .equ    01000000T   ; oVerflow
0000| 0010            Bbit        .equ    00010000T   ; Break (instead of IRQ)
0000| 0008            Dbit        .equ    00001000T   ; Decimal mode
0000| 0004            Ibit        .equ    00000100T   ; IRQ disable
0000| 0002            Zbit        .equ    00000010T   ; Zero
0000| 0001            Cbit        .equ    00000001T   ; Carry
0000| 00FA            Areg    .equ    0FA
0000| 00F9            Yreg    .equ    0F9
0000| 00F8            Xreg    .equ    0F8
0000| 00F7            SPreg   .equ    0F7
0000|
0000| 00F6            tStkP   .equ    0F6         ; save for Talk's SP
0000| 00F4            tCnt    .equ    0F4         ; count   for Talk's use
0000| 00F2            tAdr    .equ    0F2         ; address  "    "     "
0000| 00F1            tFunc   .equ    0F1         ; function code
0000| 00F0            tCKS    .equ    0F0         ; CheckSum
0000|
0000|                ;** ASCII byte defns
0000| 0002            aSTX    .equ    002         ; Start of TeXt
0000| 0003            aETX    .equ    003         ; End of TeXt
0000| 0006            aACK    .equ    006         ; ACKnowledge
0000| 0015            aNAK    .equ    015         ; Negative AcKnowledge
0000|
0000|                ;*******************************************************************************
0000|                ;*  Transaction Table definitions.  A transaction table entry is made
0000|                ;* for each SndReq and/or SndRsp request by the Host.  The REQ task creates
0000|
0000|                ;* the table entry;  the ATP task maintains is, including its destruction
0000|                ;* when it has served its useful life.  Note that all Transaction Table
0000|                ;* entries are have a "tt" prefix to distinguish them from the "at" prefix
0000|                ;* of definitions of the ATP protocol offsets.
>>>>PAGE -   9          FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000|                ;*  The definitions here are arranged so that all accesses to a tt entry
0000|                ;* can be made via "ttXXX,x", where X-reg has table locn.  In order to handle
0000|                ;* table boundary, TTbase is set to point into page such that adding ttSize
0000|                ;* to X will result in $00 when the last entry adjusted;  TTbase will then
0000|                ;* be loaded into X to handle the overflow.
```

```
0000|                 ;*  The table is split between two pages (0200/0300).  The first page
0000|                 ;* contains the ATP protocol data, the second contains our internally used
0000|                 ;* information.
0000|                 ;*****************************************************************************
0000| 000D            TTSIZE  .equ    13.         ; size of each entry
0000| 0013            TTNBR   .equ    0100/TTSIZE     ; number that fit in a page
0000| 00F7            TTSIZET .equ    TTNBR*TTSIZE        ; total size of table
0000| 0200            TTPAGE  .equ    00200           ; the page in which TTtbl is located
0000|
0000| 0209            TTBASE  .equ    TTPAGE+0100-TTSIZET ; start of table
0000| 0300            RQPAGE  .equ    TTPAGE+0100     ; RQdata
0000| 0309            RQBASE  .equ    TTBASE+0100     ; "shadow" page for lengths
0000|
0000| 0200            ttState .equ    TTPAGE          ; current state
0000| 0010            SENDREQ     .equ    010         ; new SendRequest
0000| 0011            SENDREQq    .equ    SENDREQ+1       ; SendRequest Q'd (on atTXQ)
0000| 0012            SENDREQx    .equ    SENDREQq+1      ;       "       Tx'd
0000| 0013            SENDREQw    .equ    SENDREQx+1      ;       "       waiting for response(s)
0000| 0014            SENDREQa    .equ    SENDREQw+1      ;       "       abort requested
0000| 0015            SENDREQz    .equ    SENDREQa+1      ;       "      Abort complete
0000|
0000| 0020            SENDRSP     .equ    020         ; new SendResponse
0000| 0021            SENDRSPq    .equ    SENDRSP+1       ; SendResponse Q'd (on atTXQ)
0000| 0022            SENDRSPx    .equ    SENDRSPq+1      ;       "       Tx'd
0000| 0023            SENDRSPw    .equ    SENDRSPx+1      ;       "       waiting for release (XO)
0000| 0024            SENDRSPa    .equ    SENDRSPw+1      ;       "       abort requested
0000| 0025            SENDRSPz    .equ    SENDRSPa+1      ;       "      Abort complete
0000| 0026            SENDRSPr    .equ    SENDRSPz+1      ; TRel Rcv'd while Q'd
0000| 0027            SENDRSPs    .equ    SENDRSPr+1      ;   "  processing after Tx of TResp
0000| 0028            SENDRSPn    .equ    SENDRSPs+1      ; new TReq Rcv'd                ;-B1
0000|
0000|
0000| 0030            SENDREL     .equ    030         ; send new TRel packet (for XO)
0000| 0031            SENDRELq    .equ    SENDREL+1       ; TRel Q'd
0000| 0032            SENDRELz    .equ    SENDRELq+1      ;   "  Tx'd, SENDREQ done
0000| 0033            SENDRELr    .equ    SENDRELz+1      ; temp state for Q'd SENDREQ
0000| 0034            SENDRELx    .equ    SENDRELr+1      ; TRel after Q'd SENDREQ
0000|
0000|                 ;***    the following is added to ATP, even though it supports NBP  ;-B
0000| 0040            SENDNBP     .equ    040         ; send new DDP (w/RETRY)    ;-B
0000| 0041            SENDNBPq    .equ    SENDNBP+1       ; DDP Q'd                       ;-B
0000| 0042            SENDNBPx    .equ    SENDNBPq+1      ;   "  Tx'd, timeout        ;-B
0000| 0043            SENDNBPw    .equ    SENDNBPx+1      ; timing out            ;-B
0000| 0044            SENDNBPa    .equ    SENDNBPw+1      ; SENDNBP abort         ;-B
0000| 0045            SENDNBPz    .equ    SENDNBPa+1      ; abort done            ;-B
0000| 0046            SENDNBPn    .equ    SENDNBPz+1      ; new request, temp     ;-B
0000|
0000| 0201            ttRID   .equ    ttState+1       ; RID of initial host REQuest
0000| 0202            ttLink  .equ    ttRID+1         ; link field for Tx'ing
0000| 0203            ttTmr   .equ    ttLink+1        ; dynamic timer
>>>>PAGE -  10          FILE: PCDEFS.a65  PCcard - firmware for IBM-PC AppleTalk


0000| 0203            ttBfr   .equ    ttTmr           ; holds desired buffer # during TXQ
0000|
0000|                 ;** timeout control values
0000| 0204            ttFLGs  .equ    ttTmr+1         ; Flags (initial CCI); format as atCCI
0000| 0204            ttBufN  .equ    ttFLGs          ; for SNDNBP, holds BufN    ;-B
0000|
0000| 0205            ttTmOut .equ    ttFLGS+1        ; Retry timer (SendRequest)
0000| 0205            ttRsMap .equ    ttTmOut         ; original SendResponse map
0000| 0206            ttRtrys .equ    ttTmOut+1       ; Retry count (SendRequest)
0000| 0206            ttTxMap .equ    ttRtrys         ; current SendResponse map
0000| 0207            ttDlng  .equ    ttRtrys+1       ; length of sendREQ/RSP data
0000|
0000|                 ;** info specifying DDP data
0000| 0300            ttDnet  .equ    RQPAGE          ; Dest Network # (2 bytes)
0000| 0302            ttDnode .equ    ttDnet+2        ;   "  node #
0000| 0303            ttDskt  .equ    ttDnode+1       ;   "  socket #
0000| 0304            ttSskt  .equ    ttDskt+1        ; Source socket# (from req)
0000|
0000|                 ;** info belonging to the actual ATP header
0000| 0305            ttCCI   .equ    ttSskt+1    ; request's CCI field
0000| 0306            ttBtMap .equ    ttCCI+1     ;     "     BitMap
0000| 0307            ttTID   .equ    ttBtMap+1   ;     "     Transaction ID (2 bytes)
0000| 0309            ttUser  .equ    ttTID+2     ;     "     User Data (4 bytes)
0000| 0306            ttSqNbr .equ    ttBtMap     ; response's Sequence Number
0000|
0000| 030D            ttNext  .equ    ttUser+4    ; check on TTSIZE
0000|
0000|                 .if ttNext-RQPAGE <> TTSIZE
0000|                 .endc
0000|
0000|                 ;*****************************************************************************
0000|                 ;*  definition of areas within stack area of RAM (0100 page).  Stacks
0000|                 ;* are 0180..1FF.
0000|                 ;*****************************************************************************
0000|
```

```
0000| 0100              STStbl  .equ    0100        ; actually uses 1st 040 bytes of 0100 page
0000| 0140              SKTtbl  .equ    0140        ; DDP socket registry table (32 x 8 bits)
0000| 0160              LAPtbl  .equ    0160        ; LAP protocol registry table (16 x 8 bits)
0000|
0000| 0180              STKbase .equ    0180        ; base of Task stack area
0000| 0020              STKsize .equ    020     ; size of each stack
0000|
0000|                   ;***************************************************************************
0000|                   ;** end of PCDEFS
0000|
0000|
0000|
      2 blocks for procedure code       512 words left
>>>>PAGE -  11  PCCARD    FILE:   PCcard - firmware for IBM-PC AppleTalk


0000|                       .proc   PCcard,0
Current memory available:     512
0000|
0000|                       .if fROM
0000|                       .org    0E000       ; origin for ROM based code
E000|                       .else
E000|                       .endc
E000|
E000|                       .include    PCINIT      ; Initialization code
E000|                   ;***    PCINIT, initialization code
E000|
E000| 02E0              TaskP   .word   $+2     ; dumb, f*cking assembler
E002| ****                    .word   CMDtask-1   ; initail PC's for our tasks
E004| ****                    .word   XMTtask-1
E006| ****                    .word   RCVtask-1
E008| ****                    .word   ATPtask-1
E00A|
E00A| E00A              iTbl    .equ    $       ; table base
E00A|
E00A| 0000              bTbl    .equ    $-iTbl      ;  SCC table for initialization
E00A| 09 40                  .byte    9., 040    ; reset again
E00C| 04 20                  .byte    4., 020    ; SDLC mode
E00E| 05 E1                  .byte    5., 0E1    ; no RTS, disables driver, enable DMA stuff
E010| 0A E0                  .byte   10., 0E0    ; FM0, CRC preset-1, FLAG idle, FLAG on EOM
E012| 06 00                  .byte    6., 000    ; our address
E014| 07 7E                  .byte    7., 07E    ; SDLC FLAG
E016| 0B F0                  .byte   11., 0F0    ; XTAL, Rx via DPLL, Tx via BRG
E018| 0C 06                  .byte   12., 6.     ; 16x (8-2)
E01A| 0D 00                  .byte   13., 0.
E01C| 0E C0                  .byte   14., 0C0    ; set FM mode
E01E| 0E 21                  .byte   14., 021    ; start DPLL, enable BRG
E020| 0F 08                  .byte   15., DCD    ;   DCD(PCRQ1)
E022| 10 10                  .byte   010,010     ; reset Ext int
E024| 01 09                  .byte    1., 009    ; Rx on 1st char, EXT ints
E026| 02 00                  .byte    2., 000    ; 0 IV
E028| 30                    .byte   030     ; reset Errors
E029| 38                    .byte   038     ; reset IUS
E02A|
E02A| 0020              rxRst   .equ    $-iTbl      ; reset Rx sequence
E02A| 03 C0                  .byte    3., 0C0    ; reset Rx
E02C| 0022              rxEnb   .equ    $-iTbl      ; re-enable Rx
E02C| 03 DD                  .byte    3., 0DD    ; start Rx
E02E| 20                    .byte   020     ; enable Rx
E02F| 0003              rxEnbL  .equ    $-iTbl-rxEnb
E02F| 0005
E02F|
E02F| 09 08                  .byte    9., 008    ; MIE
E031| 0027              bTblL   .equ    $-iTbl-bTbl
E031|
E031| 0027              aTbl    .equ    $-iTbl      ; table for A-port
E031| 09 80                  .byte    9., 080    ; reset
E033| 03 00                  .byte    3., 000    ; make sure of auto-enable mode
E035| 04 44                  .byte    4., 044    ; set 1-stop, ASYNC, x16 clock
E037| 0B 10                  .byte   11., 010    ; Tx via BRG
E039| 0C 96                  .byte   12., 11158. % 0100  ; gives 1 sec / (10-bit) byte
>>>>PAGE -  12  PCCARD    FILE: PCINIT.a65  PCcard - firmware for IBM-PC AppleTalk


E03B| 0D 2B                  .byte   13., 11158. / 0100
E03D| 0E 03                  .byte   14., 003    ; enable BRG, from PCLK
E03F| 0F 20                  .byte   15., CTS    ; CTS(PCRQ0)
E041| 01 03                  .byte    1., 003    ; Tx,Ext ints enabled
E043| 10 10                  .byte   010,010     ; reset Ext/Status
E045| 09 08                  .byte    9., 008    ; MIE
E047| 05 68                  .byte    5., 068    ; no DTR (IRQ to PC), enable Tx (for timing)
E049| 08 00                  .byte    8., 000    ; first xmt byte
E04B| 001A              aTblL   .equ    $-iTbl-aTbl
E04B|
E04B|                   BITS                ;** bits numbered 0..7, MSB..LSB
E04B| 80 40 20 10            .byte   080, 040, 020, 010
E04F| 08 04 02 01            .byte   008, 004, 002, 001
E053|
```

```
E053|                   atMaps              ;** # of buffers -> BtMap
E053| 00 01 03 07           .byte   000, 001, 003, 007
E057| 0F 1F 3F 7F           .byte   00F, 01F, 03F, 07F
E05B| FF                   .byte   0FF
E05C|                   atBits              ;** buffer # to BtMap
E05C| 01 02 04 08           .byte   001, 002, 004, 008
E060| 10 20 40 80           .byte   010, 020, 040, 080
E064|
E064|                   xSCCB   ;*  issue sequence of control to B-port
E064| AD 0040               lda sccCtrl     ; force sync w/ ctrl reg
E067| BD 0AE0            $1  lda iTbl,x
E06A| 8D 0040               sta sccCtrl
E06D| E8                   inx
E06E| 88                   dey
E06F| D0F6                 bne $1
E071| 60                   rts
E072|
E072|                   xSCCA   ;*  issue sequence of control to A-port
E072|
E072| AD 0240               lda sccCtrlA    ; force sync w/ ctrl reg
E075| BD 0AE0            $1  lda iTbl,x
E078| 8D 0240               sta sccCtrlA
E07B| E8                   inx
E07C| 88                   dey
E07D| D0F6                 bne $1
E07F| 60                   rts
E080|
E080|                   ;*************************************************************************
E080|                   ;*  setSTAT, routine to set IBM's readable status reg.  Note that this
E080|                   ;* will auto-magically generate an interrupt request on PC.  However, the
E080|                   ;* PC code must ensure that sufficient time has gone by before it reads the
E080|                   ;* register, since we shift data in serially.
E080|                   ;*************************************************************************
E080| 08                setSTAT php         ; save current status (for I-bit)
E081| 78                   sei         ; disable ints while dinking
E082| 8D 00C0              sta STATreg     ;  the real thing!
E085| 0A                   asl A
E086| 8D 00C0              sta STATreg
E089| 0A                   asl A
E08A| 8D 00C0              sta STATreg
E08D| 0A                   asl A
>>>>PAGE -  13  PCCARD    FILE: PCINIT.a65  PCcard - firmware for IBM-PC AppleTalk


E08E| 8D 00C0              sta STATreg
E091| 0A                   asl A       ; in case of back-to-back calls
E092| 28                   plp         ; restore I-bit
E093| 60                   rts         ;  and, return to caller
E094|
E094|                   ;*************************************************************************
E094|                   ;*  Initialization section.
E094|                   ;*************************************************************************
E094| 78                xRESET  sei         ; inhibit ints!!
E095| D8                   cld         ;  make sure of state
E096|
E096| A2 FF                ldx #0FF
E098| 9A                   txs         ; stack up hi
E099|
E099|                   ;** make sure of initial value of STATE reg
E099| A9 0F                lda #00F
E09B| 20 80E0              jsr setSTAT     ; 1st one guarantees clear
E09E| 20 80E0              jsr setSTAT     ; this one sets "initializing" state
E0A1|
E0A1|                   ;** clear RAM, for debugging
E0A1| A9 1F                lda #01F        ; all of RAM
E0A3| 85 A1               sta rP+1
E0A5| A9 00                lda #0
E0A7| 85 A0               sta rP
E0A9| A8                   tay
E0AA| 91 A0            $0  sta @rP,y
E0AC| C8                   iny
E0AD| D0FB                 bne $0
E0AF| C6 A1               dec rP+1
E0B1| 10F7                 bpl $0
E0B3|
E0B3|                       .if 1-fROM      ; for debugging only
E0B3|                       .endc
E0B3|
E0B3|                   ;*************************************************************************
E0B3|                   ;*  Initialize SCC.
E0B3|                   ;*************************************************************************
E0B3| A2 00                ldx #bTbl
E0B5| A0 27                ldy #bTblL
E0B7| 20 64E0              jsr xSCCB
E0BA|
E0BA| A2 27                ldx #aTbl           ; enable DTR, and setup ints
E0BC| A0 1A                ldy #aTblL
E0BE| 20 72E0              jsr xSCCA
```

```
E0C1|
E0C1|                       .if 1-fROM          ; for debugging $2000 only
E0C1|                       .endc
E0C1|
E0C1|                   ;*************************************************************************
E0C1|                   ;*  Initialize the free frame queue.  Note that the buffers are kept
E0C1|                   ;* track of by a number.  We only have room for 11 full buffers, so indices
E0C1|                   ;* range from 2..12.
E0C1|
E0C1|                   ;*************************************************************************
E0C1|                   initFreQ
>>>>PAGE -  14  PCCARD    FILE: PCINIT.a65  PCcard - firmware for IBM-PC AppleTalk


E0C1| A9 0C                lda #12.            ; last number
E0C3| 85 81                sta freeQtl
E0C5| A9 02                lda #2.         ; setup queue header
E0C7| 85 80                sta freeQhd
E0C9| AA                $1  tax
E0CA| 1A                   ina
E0CB| 95 80                sta freeQ,x         ; set next
E0CD| C5 81                cmp freeQtl         ; last one?
E0CF| D0F8                bne $1
E0D1|                   ;    tax                ; clear last one's next
E0D1|                   ;    stz     freeQ,x
E0D1|
E0D1| A9 09                lda #TTBASE % 0100      ; initialize ATP
E0D3| 85 E8                sta lstTTP
E0D5|
E0D5|                   ;*************************************************************************
E0D5|                   ;*  initialize the receive stuff.  Pull first buffer off of freeQ, setup
E0D5|                   ;* rxP, rP and rC.
E0D5|                   ;*************************************************************************
E0D5| A2 A4                ldx #rxP            ; pt to where we need address
E0D7| 20 ****              jsr getFree         ;  and, get one
E0DA| 20 ****              jsr setRXP
E0DD|
E0DD|                   ;*************************************************************************
E0DD|                   ;*  initialize the Task stuff.  Note that we assume that zero is the
E0DD|                   ;* default value for any values not explicitly set here;  the above clear
E0DD|                   ;* loop should guarantee that.
E0DD|                   ;*************************************************************************
E0DD|                   initTasks
E0DD| A9 01                lda #1          ; semaphore's initial count
E0DF|
E0DF| 85 74                sta DMAlock+sCnt        ;   for mutual exclusion
E0E1| 85 78                sta XMTlock+sCnt
E0E3| 85 7C                sta ATPlock+sCnt
E0E5|                   ;    stz     RCVsema+sCnt        ; RCVsema is a counter
E0E5|                   ;    stz     XMTsema+sCnt        ;  as is, XMTsema
E0E5|
E0E5|                   ;*************************************************************************
E0E5|                   ;*  setup initial entry params for our 4 tasks.
E0E5|                   ;*************************************************************************
E0E5| AD 00E0             lda TaskP           ; ptr to starting spot
E0E8| 85 02                sta t2
E0EA| AD 01E0             lda TaskP+1
E0ED| 85 03                sta t2+1
E0EF| A9 80                lda #STKbase % 0100     ; starting SP (for 4 tasks)
E0F1| A0 40                ldy #CMDtcb
E0F3|
E0F3| 85 00             $1  sta t0          ; save for looping
E0F5| 69 1F                adc #STKsize-1      ; end of stack area
E0F7| AA                   tax             ; set TopOfStack
E0F8| 9A                   txs
E0F9| A6 62                ldx TaskQtl         ; chain old last
E0FB| 84 62                sty TaskQtl         ; ours is always new last
E0FD| D0**                 bne $2          ; skip if non-empty TaskQ
E0FF|
>>>>PAGE -  15  PCCARD    FILE: PCINIT.a65  PCcard - firmware for IBM-PC AppleTalk


E0FF| 84 61                sty TaskQhd         ; only for first time!
E101| 80**                 bra $3
E103|
E0FD* 04
E103| 94 02          ~      $2  sty tNext,x         ; add ours as last's next
E101* 02
E105| B2 02             $3  lda @t2         ; plug stack (as if we did 'jsr semaP')
E107| E6 02                inc t2
E109| AA                   tax             ; reverse order
E10A| B2 02                lda @t2
E10C| E6 02                inc t2
E10E| 48                   pha
E10F| DA                   phx             ;     "     "
E110|
E110| BA                   tsx
E111| 96 03                stx tSP,y           ; keep track of stack
```

```
E113|
E113| 18                    clc             ; advance Task ptr
E114| 98                    tya
E115| 69 08                 adc #tSize
E117| A8                    tay
E118| A5 00                 lda t0          ; advance SP
E11A|
E11A| 69 20                 adc #STKsize
E11C| D0D5                 bne $1          ;  and, loop until done
E11E|
E11E| A9 80              $9  lda #DMAidle        ; show idle
E120| 85 11                 sta pcIdle
E122| 20 80E0               jsr setSTAT
E125| A9 5A                 lda #90.            ; start RTMP timer
E127| 85 1E                 sta RTMPtmr
E129|
E129| A9 01                 lda #001            ; initialize NMIflag
E12B| 85 FF                 sta NMIflg          ;  for debounce
E12D|
E12D| 4C ****               jmp Dispatch
E130|
E130|                    ;*************************************************************************
E130|                    ;*  xPC2US, diagnostic support.  Lets host write stuff to us.
E130|                    ;*************************************************************************
E130|                    xPC2US                  ;** entry from command task
E130| A9 B1                 lda #REQdata+1      ; set to do pre-fetch
E132| 20 ****               jsr setDPDC         ; set ptrs/count, wait for host
E135| 20 ****               jsr rdDMA           ; ok, copy it
E138| A9 00                 lda #0          ;  and, report success
E13A| 4C ****               jmp CMDdone
E13D|
E13D|                    ;*************************************************************************
E13D|                    ;*  xUS2PC, diagnostic support.  Lets host read stuff from us.
E13D|                    ;*************************************************************************
E13D|                    xUS2PC                  ;** entry from command task
E13D| A9 B0                 lda #REQdata        ; no pre-fetch
E13F| 20 ****               jsr setDPDC         ; set ptrs/count, wait for host
E142| 20 ****               jsr wrDMA           ; ok, copy it
E145| A9 00                 lda #0          ;  and, report success
>>>>PAGE -  16  PCCARD    FILE: PCINIT.a65  PCcard - firmware for IBM-PC AppleTalk


E147|
E147| 4C ****               jmp CMDdone
E14A|
E14A|                    ;*************************************************************************
E14A|                    ;*  setDPDC, common code for xPC2US/xUS2PC to read DMA specs, wait for
E14A|                    ;* user to show its ready.
E14A|                    ;*************************************************************************
E140* 4AE1
E133* 4AE1
E14A|                    setDPDC                 ;** entry from xPC2US/xUS2PC
E14A| 48                    pha             ; save waitDMA code
E14B| AD 0080               lda DMAreg          ; setup DMA transfer. (4 bytes)
E14E| 85 18                 sta dP
E150| AD 0080               lda DMAreg
E153| 85 19                 sta dP+1
E155| AD 0080               lda DMAreg
E158| 85 1A                 sta dC
E15A| AD 0080               lda DMAreg
E15D| 85 1B                 sta dC+1
E15F|
E15F| 68                    pla             ; tell host we are ready
E160| 20 ****               jsr waitDMA
E163|
E163| 60                    rts             ;  and, let caller finish up
E164|
E164|                    ;** end of PCINIT.
E164|
E164|
E164|                        .include    PCEXEC      ; Multi-tasking kernel
E164|                    ;*  PCEXEC, executive of the PC AppleBus card.
E164|                    ;*************************************************************************
E164|                    ;*  These routines comprise a small "multi-tasking" executive which
E164|                    ;* allows multiple commands to be in progress (from the PC) in a somewhat
E164|                    ;* controlled fashion.  The resources which require exclusive access (the
E164|                    ;* DMA channel and the Transmitter) are controlled via semaphores.
E164|                    ;*************************************************************************
E164|
E164|                    ;*************************************************************************
E164|                    ;*  Wait, wait for a specified signal (A-reg) to occur.  If the flag is
E164|                    ;* currently set in our tcb, then treat as a nop.
E164|                    ;*************************************************************************
E164| 08                 Wait    php         ; save I
E165| 78                    sei         ;  and, inhibit
E166| A6 60                 ldx theTask     ; our tcb
E168| 34 00                 bit tFlags,x    ; any matching
E16A| D0**                 bne $9      ;  yes, it already happened, don't wait
```

```
E16C| 95 01               sta tWaits,x    ; nope, show the ones we're interested in
E16E| 28                  plp         ; restore I
E16F| 4C ****             jmp yield1      ;  and, stop until we get it
E172|
E16A* 06
E172| 28               $9  plp         ; just return to caller
E173|
E173| 60                  rts
E174|
>>>>PAGE -  17  PCCARD    FILE: PCEXEC.a65  PCcard - firmware for IBM-PC AppleTalk


E174|                   ;*************************************************************************
E174|                   ;*  Signal, send a signal to a task;  if that task is indeed waiting for
E174|                   ;* the desired flag, make it ready.  X has tcb address, A has signal bits.
E174|                   ;*************************************************************************
E174| 08                Signal  php         ; save I
E175| 78                  sei         ;  and, inhibit
E176| 34 01               bit tWaits,x    ; is it waiting for this?
E178| F0**                beq $9      ;  no, merely post
E17A| 34 00               bit tFlags,x    ; else, is it alREADY?
E17C| 30**                bmi $9      ; yeah, just post
E17E|
E17E|                   ;** task is not running, and a flag matches, make it ready and queue it.
E17E| 48                  pha         ; save flags for ORring
E17F| 8A                  txa         ; enqueue it
E180| A6 62               ldx TaskQtl
E182| 85 62               sta TaskQtl
E184| D0**                bne $1
E186| 85 61               sta TaskQhd     ; if queue is empty, make it head also
E188| 80**                bra $2
E184* 04
E18A| 95 02           $1  sta tNext,x     ; else, ours is next of last's
E188* 02
E18C| AA               $2  tax
E18D| 74 02               stz tNext,x     ; make sure of our next
E18F| 68                  pla         ; restore flags
E190| 09 80               ora #tReady     ; show alREADY
E192|
E17C* 14
E178* 18
E192| 15 00           $9  ora tFlags,x    ; accumulate flags
E194| 95 00               sta tFlags,x
E196| 28                  plp         ; restore I
E197| 60                  rts         ;  and, leave
E198|
E198|                   ;*************************************************************************
E198|                   ;*  Semaphore management routines.  "Classical" semaphores, nothing very
E198|
E198|                   ;* fancy.  Note that X contains the address of the semaphore (obviously in
E198|                   ;* zero-page).  All regs destoryed.
E198|                   ;*************************************************************************
E198|
E198|                   semaV   ;** the classical V operation.
E198| 08                  php         ; save current I
E199| 78                  sei         ; force atomicity
E19A| 74 03               stz sOwn,x      ; show no owner now
E19C| F6 00               inc sCnt,x      ; up the count
E19E| 30**                bmi $1      ;  if <=0, schedule one
E1A0| D0**                bne semaX       ; common exit if >0
E1A2|
E19E* 02
E1A2| B4 01           $1  ldy sQhd,x      ; unlink 1st on queue
E1A4| 94 03               sty sOwn,x      ;  indicate our new owner
E1A6| B9 02 00            lda tNext,y     ; its nexts
E1A9| 95 01               sta sQhd,x      ;   is new head
E1AB| D0**                bne $2
>>>>PAGE -  18  PCCARD    FILE: PCEXEC.a65  PCcard - firmware for IBM-PC AppleTalk


E1AD| 95 02                 sta sQtl,x      ; if no more, make Q empty
E1AF|
E1AB* 02
E1AF| 28               $2  plp         ; we can allow ints for a while
E1B0| 98                    tya         ; copy dequeued task to param reg
E1B1| 08                    php
E1B2| 78                    sei         ; OK, prevent again
E1B3|
E1B3|                   ;** enq the task to taskQ
E1B3| A6 62                 ldx TaskQtl     ; append to end
E1B5| F0**                  beq $3      ; skip if currently empty
E1B7| 95 02                 sta tNext,x     ;  else, add as next of last
E1B9| 80**                  bra $4
E1B5* 04
E1BB| 85 61             $3  sta TaskQhd     ; if empty, make it first
E1B9* 02
E1BD| 85 62             $4  sta TaskQtl     ;  as well as last
E1BF| AA                    tax         ; make sure of "end"
```

```
E1C0| 74 02                stz tNext,x
E1C2| A9 80                lda #tREADY     ; show ready
E1C4| 15 00                ora tFlags,x
E1C6| 95 00                sta tFlags,x
E1C8| 74 01
E1CA|
E1A0* 28
E1CA| 28                  semaX   plp         ; allow ints (maybe),
E1CB| 60                      rts         ;  on way out
E1CC|
E1CC|                      semaP   ;** classical P operation.
E1CC| 08                      php
E1CD| 78                      sei         ; enforce atomicity
E1CE| D6 00                   dec sCnt,x      ; down the count
E1D0| 30**                    bmi $0
E1D2| A5 60                   lda theTask     ; show who owns it
E1D4| 95 03                   sta sOwn,x
E1D6| 80F2                    bra semaX       ; common exit if ok
E1D8|
E1D0* 06
E1D8| A5 60                $0  lda theTask     ; ok, who are we?
E1DA| B4 02                   ldy sQtl,x      ; get current last
E1DC| F0**                    beq $1      ; there is no one, skip linking
E1DE|
E1DE| 99 02 00                sta tNext,y     ; add ours to last's next
E1E1| 80**                    bra $2      ; merge
E1E3|
E1DC* 05
E1E3| 95 01                $1  sta sQhd,x      ; if no one on, make ours first
E1E1* 02
E1E5| 95 02                $2  sta sQtl,x      ;  always last
E1E7| AA                      tax         ; task's address
E1E8| 74 02                   stz tNext,x     ; make sure we're "last"
E1EA| A9 01                   lda #sSEMA      ; show waiting for semaphore
E1EC| 95 01                   sta tWaits,x
E1EE| 49 7F                   eor #07F        ; mask this one out
E1F0| 35 00                   and tFlags,x
>>>>PAGE -  19  PCCARD    FILE: PCEXEC.a65  PCcard - firmware for IBM-PC AppleTalk


E1F2| 95 00                   sta tFlags,x
E1F4|
E1F4| 28                      plp         ; restore I-bit
E1F5| 80**                    bra yield1      ;  and, give up CPU
E1F7|
E1F7|                      ;*************************************************************************
E1F7|                      ;*  YIELD, the entry to voluntarily give up control.  If theTask is not
E1F7|                      ;* null, the task is requeued on to the end of the ready task queue.
E1F7|                      ;*************************************************************************
E1F7|                      Yield
E1F7| 78                      sei         ; inhibit while dinking
E1F8|
E1F8| A5 60                   lda theTask     ; is one active?
E1FA| F0**                    beq Dispatch    ;  nope, simply schedule another
E1FC|
E1FC|                      ;** enq the task to taskQ
E1FC| A6 62                   ldx TaskQtl     ; append to end
E1FE| F0**                    beq $1      ; skip if currently empty
E200| 95 02                   sta tNext,x     ;  else, add as next of last
E202| 80**                    bra $2
E1FE* 04
E204| 85 61                $1  sta TaskQhd     ; if empty, make it first
E202* 02
E206| 85 62                $2  sta TaskQtl     ;  as well as last
E208| AA                      tax         ; make sure of "end"
E209| 74 02                   stz tNext,x
E20B|
E1F5* 14
E170* 0BE2
E20B|                      yield1  ;** common point to save state of current task
E20B| A4 60                   ldy theTask     ; save our stack base
E20D| BA                      tsx         ;  save its stack ptr
E20E| 96 03                   stx tSP,y
E210| 64 60                   stz theTask
E212|
E212|                      ;*************************************************************************
E212|                      ;*  DISPATCH, the main "exec" loop.  We wait until the taskQ has a task
E212|                      ;* to dispatch to.  Note that interrupts (particularly, received packets
E212|                      ;* and/or Function requests from the PC) will place tasks onto the taskQ.
E212|                      ;*************************************************************************
E1FA* 16
E12E* 12E2
E212|                      Dispatch
E212| 58                      cli         ; this loop must! be interruptable
E213| A9 20                $1  lda #fREQ       ; request loop flag
E215| 14 10                   trb pcFlags     ; look for request from PC
E217| F0**                    beq $2
E219|
```

```
E219|                   ;** we have seen a request from PC host, issue a task, if any free
E219|
E219| A2 70                 ldx #REQsema
E21B| 20 98E1               jsr semaV       ; inform CMDtask
E21E|
E217* 05
E21E| A4 61             $2  ldy TaskQhd     ; is there one?
>>>>PAGE -  20  PCCARD    FILE: PCEXEC.a65  PCcard - firmware for IBM-PC AppleTalk


E220| F0F1                 beq $1      ;  nope, how dull....
E222|
E222| 78                   sei         ; while launching, don't mess..
E223| B9 02 00             lda tNext,y     ; remove from queue
E226| D0**                 bne $3
E228| 85 62                sta TaskQtl     ; if null, mark that fact
E226* 02
E22A| 85 61             $3  sta TaskQhd     ;  always, new head
E22C|
E22C| B6 03                 ldx tSP,y       ; ok, switch the stack
E22E| 9A                   txs
E22F| 84 60                sty theTask     ; show who's running
E231| 58                   cli         ; user is interruptable
E232| 60                   rts         ;  and, off to it
E233|
E233|                   ;*****************************************************************************
E233|                   ;*  getFree, obtain a buffer from freeQ, and place its converted address
E233|                   ;* into area pointed to by X.
E233|                   ;*****************************************************************************
E0D8* 33E2
E233| 08               getFree php         ; save I-bit
E234| 78                   sei         ;  then, inhibit
E235| A4 80                ldy freeQhd     ; get current header
E237| F0**                 beq $2      ; if empty, forget queue update
E239| B9 80 00             lda freeQ,y     ; get its next
E23C| D0**                 bne $1      ; skip if not last
E23E| 85 81                sta freeQtl     ; else, update to show now empty
E23C* 02
E240| 85 80             $1  sta freeQhd     ; always, new head
E242|
E237* 09
E242| 28               $2  plp         ; get original I-bit back
E243| 98                   tya         ; get buffer#
E244|
E244|                   cvtBufN ;** entry to convert buffer # to address (index in X)
E244|
E244| 48                   pha         ; save buffer number
E245| 95 01                sta 1,x     ; save upper part
E247| 0A                   asl A       ; convert it
E248| 0A                   asl A
E249| 75 01                adc 1,x     ; by *5
E24B| 95 01                sta 1,x
E24D| A9 00                lda #0
E24F| 95 00                sta 0,x     ; setup for shift
E251| 56 01                lsr 1,x     ; adjust it
E253| F0**                 beq $9      ; skip if empty value
E255| 76 00                ror 0,x
E257| D6 01                dec 1,x
E253* 04
E259| 68               $9  pla         ; restore buffer #
E25A| 60                   rts         ; and, that's it...
E25B|
E25B|                   ;*****************************************************************************
E25B|                   ;*  setRXP, common code to update working rP/rC from rxP.  A-reg has
E25B|                   ;* been loaded (Z-bit s.b. correct) w/ new buffer number.
>>>>PAGE -  21  PCCARD    FILE: PCEXEC.a65  PCcard - firmware for IBM-PC AppleTalk


E25B|                   ;*****************************************************************************
E0DB* 5BE2
E25B| 08               setRXP  php         ; save I-bit
E25C| 78                   sei         ; inhibit while updating rP/rC
E25D| 85 A6                sta rxBufN      ;   save where it is
E25F| F0**                 beq $9      ; skip nonsense if none
E261| A5 A4                lda rxP     ; setup working values
E263| 85 A0                sta rP
E265| A5 A5                lda rxP+1
E267| 85 A1                sta rP+1
E269| A9 9F                lda #~<rxCsz % 0100>    ; load w/ complement of size
E26B| 85 A2                sta rC
E26D| A9 FD                lda #~<rxCsz / 0100>
E26F| 85 A3                sta rC+1
E25F* 10
E271| 28               $9  plp         ; restore old
E272| 60                   rts         ;  and, exit
E273|
E273|                   ;*****************************************************************************
E273|                   ;*  putFree, place buffer back onto free buffer queue.  Note that due to
```

```
E273|
E273|                   ;* the way in which freeQ is organized, the 'sta freeQ,x' will set freeQhd
E273|                   ;* for the case where the queue is currently empty.
E273|                   ;*************************************************************************
E273| 08                putFree php         ; inihibit
E274| 78                    sei
E275| AA                    tax         ; clear next
E276| 74 80                 stz freeQ,x
E278| A6 81                 ldx freeQtl     ; get current last
E27A| 95 80                 sta freeQ,x     ;  and, make ours its next.
E27C| 85 81                 sta freeQtl     ; update ours as last
E27E| 28                    plp
E27F| 60                    rts         ;  and, we're done
E280|
E280|                   ;*************************************************************************
E280|                   ;*  putXmit, add a buffer (in A) to the XMTtask queue;  schedule XMTtask.
E280|                   ;*************************************************************************
E280|                   putXmit                 ;** called by Tx'ers
E280| A6 25                 ldx xmitQtl         ; add to tail
E282| 85 25                 sta xmitQtl
E284| D0**                  bne $1          ; if empty
E286| 85 24                 sta xmitQhd         ;  our is only one
E288| 80**                  bra $2
E284* 04
E28A| 95 80             $1  sta freeQ,x         ; else, queue ours to last
E288* 02
E28C| AA               $2   tax
E28D| 74 80                 stz freeQ,x         ; and plug our next
E28F|
E28F|                   ;** enqueue to XMTtask, wait for completion
E28F| A2 20                 ldx #XMTsema        ; let XMTtask know about it
E291| 20 98E1               jsr semaV
E294| 60                    rts
E295|
E295|                   ;*************************************************************************
>>>>PAGE -  22  PCCARD    FILE: PCEXEC.a65  PCcard - firmware for IBM-PC AppleTalk


E295|
E295|                   ;*  SCCint, initial entry from any SCC interrupt source.  Figure out
E295|                   ;* the cause and vector to the appropriate handler.  SCCintX is a secondary
E295|                   ;* entry to re-examine interrupts for overlapping service requests.
E295|                   ;*************************************************************************
E295| A9 00             Atbe    lda #000            ; start transmitter again
E297| 8D 0340               sta sccDataA
E29A|
E29A|                   ;** 1 second timer overflowed, we have a full second!!
E29A| E6 1C                 inc TMRsecs         ; incr 16-bit counter
E29C| D0**                  bne $1
E29E| E6 1D                 inc TMRsecs+1
E2A0|
E29C* 02
E2A0| C6 1E             $1  dec RTMPtmr         ; check on RTMP invalidation
E2A2| 10**                  bpl SCCintX
E2A4| 64 0B                 stz aBridge         ; invaliditate the bridge #
E2A6| A9 5A                 lda #90.            ;  and, reload timer
E2A8| 85 1E                 sta RTMPtmr
E2AA| 80**                  bra SCCintX
E2AC|
E2AC|                   ;****   SCCint, the 1st actual interrupt entry point.
E2AC| 48               SCCint  pha             ; save reg to use
E2AD|
E2AD|                   ;***    SCCintX, return point to look for more interrupts.
E2AA* 01
E2A2* 09
E2AD| A9 03             SCCintX lda #3.         ; read RR3 (SCC's IP bits)
E2AF| 8D 0240               sta sccCtrlA
E2B2| A9 3F                 lda #03F
E2B4| 2D 0240               and sccCtrlA
E2B7| D0**                  bne $1          ; some present, look at 'em
E2B9| 68                    pla             ;  else, leave
E2BA| 40                    rti
E2BB|
E2BB|                   ;***    at least 1 IP bit is present;  find out which one.
E2B7* 02
E2BB| 89 04             $1  bit #004            ; special case for RCA-B?
E2BD| F0**                  beq $2
E2BF| 4C ****               jmp rxIntr          ;  yes, handle elsewhere
E2C2|
E2C2|
E2BD* 03
E2C2| 89 01             $2  bit #001            ; EXT-B?
E2C4| D0**                  bne Bext
E2C6| 89 08                 bit #008            ; EXT-A?
E2C8| F0CB                  beq Atbe            ; if not, must be TBE-A
E2CA|
E2CA|                   ;** assume EXT/STATUS if not TBE
E2CA| AD 0240           Aext    lda sccCtrlA        ; ok, load our status
```

```
E2CD| 29 20                 and #CTS            ;  aka, fREQ
E2CF| 05 10                 ora pcFlags
E2D1| 85 10                 sta pcFlags
E2D3| A9 10                 lda #010            ; Reset EXT
E2D5| 8D 0240               sta sccCtrlA
>>>>PAGE -  23  PCCARD    FILE: PCEXEC.a65  PCcard - firmware for IBM-PC AppleTalk


E2D8| 80D3                 bra SCCintX         ;  and, re-examine
E2DA|
E2C4* 14
E2DA| AD 0040           Bext    lda sccCtrl         ; ok, load our status
E2DD| 29 08                 and #DCD            ;  aka, fDMA
E2DF| 05 10                 ora pcFlags
E2E1| 85 10                 sta pcFlags
E2E3| A9 10                 lda #010            ; issue EXT reset
E2E5| 8D 0040               sta sccCtrl
E2E8| 80C3                 bra SCCintX
E2EA|
E2EA|                   ;*************************************************************************
E2EA|                   ;** end of pcEXEC
E2EA|
E2EA|                       .include    PCCMDS      ; Host command processing
E2EA|                   ;** PCCMDS, Command execution routines from PC.
E2EA|
E2EA|                   ;*************************************************************************
E2EA|                   ;*  cTbl, the command table which is searched by xPCcmd when a new
E2EA|                   ;* command has been processed.  Note, the address of the command is -1 due
E2EA|                   ;* to the manner in which RTS works.
E2EA|                   ;*************************************************************************
E2EA|                   cTbl    ;* base of table
E2EA|                   ;    .byte   010             ; init LAP (and CARD) (special check)
E2EA|                   ;    .word   xINITLAP-1
E2EA| 01                      .byte   001         ; host -> card
E2EB| 2FE1                    .word   xPC2US-1
E2ED| 02                      .byte   002         ; card -> host
E2EE| 3CE1                    .word   xUS2PC-1
E2F0| 04                      .byte   004         ; read SysFlg
E2F1| ****                    .word   xRDSFLG-1
E2F3| 05                      .byte   005         ; write SysFlg
E2F4| ****                    .word   xWRSFLG-1
E2F6| 08                      .byte   008         ; BRIDGE info
E2F7| ****                    .word   xBRIDGE-1
E2F9| 11                      .byte   011         ; ADD LAP type
E2FA| ****                    .word   xADDLAP-1
E2FC| 12                      .byte   012         ; DELete LAP type
E2FD| ****                    .word   xDELLAP-1
E2FF| 13                      .byte   013         ; Transmit LAP pkt
E300| ****                    .word   xTXLAP-1
E302| 21                      .byte   021         ; DDP OpenSocket
E303| ****                    .word   xOPNSKT-1
E305| 22                      .byte   022         ; DDP CloseSocket
E306| ****                    .word   xCLSSKT-1
E308| 23                      .byte   023         ; Tx DDP (short/extended)
E309| ****                    .word   xTXDDP-1
E30B| 25                      .byte   025         ; Tx DDP (short/extended w/ CKS)
E30C|
E30C| ****                    .word   xTXDDP-1
E30E| 27                      .byte   027         ; Tx DDP (w/ retries)
E30F| ****                    .word   xSNDNBP-1
E311| 2F                      .byte   02F         ; Abort xTXDDP w/retries
E312| ****                    .word   xABTNBP-1
>>>>PAGE -  24  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E314| 31                      .byte   031         ; SendRequest
E315| ****                    .word   xSNDREQ-1
E317| 32                      .byte   032         ; SendResponse
E318| ****                    .word   xSNDRSP-1
E31A| 33                      .byte   033         ; AddResponse
E31B| ****                    .word   xADDRSP-1
E31D| 39                      .byte   039         ; Abort a SendRequest
E31E| ****                    .word   xABTREQ-1
E320| 3A                      .byte   03A         ; Abort a SendResponse (usually XO)
E321| ****                    .word   xABTRSP-1
E323| 0039                cTblL   .equ    $-cTbl          ; length of table
E323|
E323|                   ;*************************************************************************
E323|                   ;*  postSTS, common subroutine to add entry to STStbl.  A has RID,
E323|                   ;* Y has STATUS.
E323|                   ;*************************************************************************
E323|                   postSTS                 ;** called from elsewhere!!
E323| A6 8F                   ldx STSQtl          ; make sure of room
E325| 9D 0001                 sta STStbl,x
E328| E8                      inx
E329| 98                      tya
E32A| 9D 0001                 sta STStbl,x
```

```
E32D| E8                    inx
E32E| 8A                    txa
E32F| 29 3F                 and #03F            ; handle wraparound
E331| C5 8E                 cmp STSQhd          ; overflow?
E333| F0**                  beq $1          ;  yes, don't update tail
E335| 85 8F                 sta STSQtl          ; else, we're ok..
E337|
E333* 02
E337| A9 90              $1  lda #STSrdy
E339| 85 11                 sta pcIdle
E33B|
E33B|                   ;** see if we can post STSrdy to Host now..
E33B| A5 77                 lda DMAlock+sOwn        ; anyone own DMA?
E33D| D0**                  bne $9          ;  if so, don't screw w/STATreg
E33F|
E33F| A5 11                 lda pcIdle          ; else, inform user
E341| 20 80E0               jsr setSTAT
E344|
E33D* 05
E344| 60                  $9  rts
E345|
E345|                   ;*************************************************************************
E345|                   ;*  CMDstat, get here to post ending status of current command.  Note
E345|                   ;* that we assume that STSQ can never overflow!!!!
E345|                   ;*************************************************************************
E148* 45E3
E13B* 45E3
E345| 85 45              CMDdone sta CMDRslt         ; save result
E347| A2 74                 ldx #DMAlock        ; free DMA
E349| 20 98E1               jsr semaV
E34C|
E34C| A5 47              CMDstat lda CMDRID          ; RID
E34E| F0**                  beq $1          ; 0 -> no status now!!
>>>>PAGE -  25  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E350| A4 45                 ldy CMDRslt
E352| 20 23E3               jsr postSTS         ; show user
E355|
E355|                   ;** check if we should setSTAT for caller
E34E* 05
E355| A5 77              $1  lda DMAlock+sOwn        ; anyone own DMA?
E357| D0**                  bne CMDtask         ;  if so, don't screw w/STATreg
E359|
E359| A5 11                 lda pcIdle          ; else, inform user
E35B| 20 80E0               jsr setSTAT
E35E|
E35E|                   ;*************************************************************************
E35E|                   ;*  CMDtask, command execution master loop.  This task waits until
E35E|                   ;* REQsema gets cranked, as the result of a pcREQ seen.
E35E|                   ;*************************************************************************
E35E|
E357* 05
E002* 5DE3
E35E| 58                  CMDtask cli         ; make sure ints can come in
E35F| A2 70                 ldx #REQsema    ; wait for something to do
E361|
E361| 20 CCE1               jsr semaP
E364|
E364| A2 74                 ldx #DMAlock    ; then, ask for use of DMA
E366| 20 CCE1               jsr semaP
E369|
E369|                   ;** fetch the command from PC (fixed length)
E369| A9 A1                 lda #REQinfo+1  ; show what we want on DMA
E36B| 20 ****               jsr waitDMA     ;  and, wait
E36E|
E36E| AD 0080               lda DMAreg      ; fetch REQcode
E371| 85 12                 sta pcREQ       ; save for search loop
E373| AD 0080               lda DMAreg      ;  and RID
E376| 85 47                 sta CMDRID
E378|
E378|                   ;***    save last 8 commands in CMDhst
E378| A6 2F                 ldx CMDhstP     ; current "next"
E37A| A5 12                 lda pcREQ
E37C| 9D 801F               sta CMDhst,x
E37F| E8                    inx
E380| A5 47                 lda CMDRID
E382| 9D 801F               sta CMDhst,x
E385| E8                    inx
E386| 8A                    txa         ; update ptr
E387| 29 3E                 and #03E        ; (modulo 40)
E389| 85 2F                 sta CMDhstP
E38B|
E38B|                   ;*************************************************************************
E38B|                   ;*  to make sure that we always can accept STATUS commands, we must be
E38B|                   ;* sure that one task always is available for non-STATUS.
E38B|                   ;*************************************************************************
E38B| A5 12                 lda pcREQ       ; get request
```

```
E38D| F0**                beq xSTATUS     ;  faster processing for STATUS request
E38F|
E38F|                   ;** decode the command type, backwords search of command table
>>>>PAGE -  26  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E38F| A2 39             $1  ldx #cTblL      ; index (and loop counter)
E391| C9 10                 cmp #010        ; xINITLAP request?
E393| D0**                 bne $2
E395| 4C ****               jmp xINITLAP    ;  yes, off to it now
E398|
E393* 03
E398| 29 F8             $2  and #0F8        ; special range?
E39A|
E39A| F0**                 beq $21     ;  yes, allow it
E39C| 24 13                 bit myFlags     ; no, check if we've been initialized
E39E| 10**                 bpl xPCerr      ;  no, we can't do anything 'til we are
E3A0|
E39A* 04
E3A0| A5 12             $21 lda pcREQ       ; re-load it
E3A2|
E3A2| DD EAE2           $3  cmp Ctbl,x      ; is this it?
E3A5| F0**                 beq goCmd       ;   yep...
E3A7| CA                   dex         ; else, count down
E3A8| CA                   dex
E3A9| CA                   dex
E3AA| 10F6                 bpl $3      ;  loop if still more left
E3AC|
E3AC|                   ;** illegal command
E39E* 0C
E3AC| A9 FF             xPCerr  lda #REQerr     ; set error status for return to CMDtask
E3AE| 4C 45E3               jmp CMDdone
E3B1|
E3B1|                   ;** found command, off to it
E3A5* 0A
E3B1| 64 45             goCmd   stz CMDrslt     ; set default result
E3B3| BD ECE2               lda cTbl+2,x    ; fetch the address (-1, due to RTS)
E3B6| 48                   pha
E3B7| BD EBE2               lda cTbl+1,x
E3BA| 48                   pha
E3BB| 60                   rts
E3BC|
E3BC|                   ;*************************************************************************
E3BC|                   ;*  xSTATUS, code for status request from PC.  If status is available,
E3BC|                   ;* send to Host, else return 0.
E3BC|                   ;*************************************************************************
E38D* 2D
E3BC| A9 B0             xSTATUS lda #REQdata        ; tell host
E3BE| 20 ****               jsr waitDMA
E3C1|
E3C1| A6 8E                 ldx STSQhd          ; any status ready?
E3C3| E4 8F                 cpx STSQtl
E3C5| F0**                 beq $0          ; nope, return null
E3C7| BD 0001               lda STStbl,x        ; else, report it
E3CA| E8                   inx
E3CB| 8D 0080               sta DMAreg          ; RID
E3CE| BD 0001               lda STStbl,x
E3D1| E8                   inx
E3D2| 8D 0080               sta DMAreg          ; Rslt
E3D5| 8A                   txa
E3D6| 29 3F                 and #03F            ; mask around the end
>>>>PAGE -  27  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E3D8|
E3D8| 85 8E                 sta STSQhd
E3DA|
E3DA| A2 80             $1  ldx #DMAidle        ; assume empty
E3DC| C5 8F                 cmp STSQtl          ; empty now?
E3DE| F0**                 beq $2
E3E0| A2 90                 ldx #STSrdy         ; else, show more
E3DE* 02
E3E2| 8A                $2  txa
E3E3|
E3E3| 85 11             $3  sta pcIdle          ; save for later
E3E5| 20 80E0               jsr setSTAT         ; do it
E3E8| A2 74                 ldx #DMAlock        ;  and, free DMA
E3EA| 20 98E1               jsr semaV
E3ED| 4C 5EE3               jmp CMDtask         ; back for more
E3F0|
E3C5* 29
E3F0| 9C 0080           $0  stz DMAreg          ; return 0
E3F3| A9 80                 lda #DMAidle        ; delay for some
E3F5| 80EC                 bra $3          ;  and share
E3F7|
E3F7|                   ;*************************************************************************
E3F7|                   ;*  xWRSFLG, set SysFlgs.  Note: the value we set gets returned, since
E3F7|                   ;* we fall thru into RDSFLG code!!
```

```
E3F7|                   ;*  xRDSFLG, return current SysFlgs stuff.  Note: the actual value gets
E3F7|                   ;* returned as "status" for this request.
E3F7|                   ;*************************************************************************
E2F4* F6E3
E3F7| AD 0080           xWRSFLG lda DMAreg          ; get the new value
E3FA| 85 30                 sta SysFlgs         ;  and, stash it
E3FC|
E2F1* FBE3
E3FC| A5 30             xRDSFLG lda SysFlgs         ; get current values (as STATUS)
E3FE| 4C 45E3               jmp CMDdone         ;  and, merge
E401|
E401|                   ;*************************************************************************
E401|                   ;*  xBRIDGE, return current BRIDGE stuff.
E401|                   ;*************************************************************************
E401|
E2F7* 00E4
E401| A9 B0             xBRIDGE lda #REQdata        ; tell user about it
E403| 20 ****                jsr waitDMA
E406|
E406| A5 0B                   lda aBridge         ; get current values
E408| 8D 0080                 sta DMAreg
E40B| A5 0C                   lda ThisNet
E40D| 8D 0080                 sta DMAreg
E410| A5 0D                   lda ThisNet+1
E412| 8D 0080                 sta DMAreg
E415|
E415| 64 47                   stz CMDRID          ; show no status
E417| 4C 45E3                 jmp CMDdone         ;  and, merge
E41A|
E41A|                   ;*************************************************************************
E41A|                   ;*  waitDMA, routine to be called by two-phase commands.  We issue the
>>>>PAGE -  28  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E41A|                   ;* setSTAT for the first part of the command and wait for the fDMA bit to
E41A|                   ;* be set for second phase, whereupon we return to our caller.  A-reg has
E41A|                   ;* the code to send to PC, which indicates what we're waiting for.  Low-bit
E41A|                   ;* of A indicates a read, so pre-fetch to "prime" the reg.
E41A|                   ;*************************************************************************
E41A|                   waitDMA1    ;** intial entry to set STATUS
E41A| 48                      pha             ; save RD flags
E41B| A9 08                   lda #fDMA           ; clear any old DMA flag
E41D| 14 10                   trb pcFlags
E41F| 68                      pla             ; get function back
E420| 20 80E0                jsr setSTAT         ; send, w/ interrupt
E423| 60                      rts
E424|
E404* 24E4
E3BF* 24E4
E36C* 24E4
E424| 48             waitDMA pha             ; save it
E425| 20 1AE4               jsr waitDMA1        ; for full treatment, call one
E428| 68                      pla             ; restore it
E429|                                   ;  then, fall thru
E429|
E429|                   waitDMA2    ;** re-entry to wait for fDMA
E429| 48                      pha             ; save function (low-bit counts)
E42A| 20 F7E1           $1   jsr Yield           ; "wait" a while
E42D| A9 08                   lda #fDMA
E42F| 14 10                   trb pcFlags         ;  wait for the bit from PC to clear
E431| F0F7                   beq $1
E433|
E433| 68                      pla             ; get req back
E434| 6A                      ror A           ; shift to check bit-0
E435| 90**                    bcc $9          ;  skip if not
E437| AD 0080                 lda DMAreg          ; else, do the pre-fetch
E43A|
E435* 03
E43A| 60               $9   rts             ;   then, return
E43B|
E43B|                   ;*************************************************************************
E43B|                   ;*  freeDMA, the tail-end code for a process which has allocated DMA and
E43B|                   ;* wishes to send "ending" status to Host, making sure the Host sees it.  We
E43B|                   ;* issue the status and wait for the DMA request bit to go to 0;  this
E43B|                   ;* indicates that the Host is acknowledging the completion.
E43B|                   ;*************************************************************************
E43B| A5 11             freeDMA lda pcIdle          ; always show current idle value
E43D| 20 80E0                jsr setSTAT         ; issue the status update
E440|
E440| 20 F7E1           $1   jsr Yield           ; give others a chance
E443| A9 08                   lda #fDMA           ; then, check it
E445| 2C 0040                 bit SCCctrl         ; set?
E448| D0F6                   bne $1          ;  yep, Host still doesn't know
E44A|
E44A|                   ;** having seen Host acknowledge it, free up DMA
E44A|
E44A| A2 74                   ldx #DMAlock
E44C| 20 98E1                jsr semaV
```

```
E44F| 60                    rts
>>>>PAGE -  29  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E450|
E450|                   ;*************************************************************************
E450|                   ;*  rdDMA, read a block of data from DMAreg into area defined by dP/dC.
E450|                   ;* To increase speed, we do them in chunks of 4, then remainder.
E450|                   ;*************************************************************************
E450| A0 00             rdDMA   ldy #0          ; starting offset
E452| A5 1A                 lda dC          ; pickup low-byte of count
E454| 66 1B                 ror dC+1            ; shift hi
E456| 6A                   ror A           ;  into lo
E457| 66 1B                 ror dC+1
E459| 6A                   ror A           ;  for 2 bits
E45A| F0**                 beq $3          ; skip to remainder case if no 4x
E45C| AA                   tax             ;  else, copy for loop's usage
E45D|
E45D| AD 0080           $1  lda DMAreg          ; copy 4 fast...
E460| 91 18                 sta @dP,y
E462| C8                   iny
E463| AD 0080               lda DMAreg
E466| 91 18                 sta @dP,y
E468| C8                   iny
E469| AD 0080               lda DMAreg
E46C| 91 18                 sta @dP,y
E46E| C8                   iny
E46F| AD 0080               lda DMAreg
E472| 91 18                 sta @dP,y
E474| C8                   iny
E475| D0**                 bne $2          ; we only test after every 4 bytes
E477| E6 19                 inc dP+1
E475* 02
E479| CA                 $2  dex             ; down-count of /4
E47A| D0E1                 bne $1
E47C|
E47C|                   ;** handle remainder 0..3 bytes here
E45A* 20
E47C| A5 1A             $3  lda dC          ; pickup low 3 bytes
E47E| 29 03                 and #003            ; use only 2 bits
E480| F0**                 beq $9          ; yeah, all done
E482| AA                   tax             ;  else, copy for loop
E483|
E483| AD 0080           $4  lda DMAreg          ; get it
E486| 91 18                 sta @dP,y           ;  stash it
E488| C8                   iny
E489| CA                   dex             ; down count
E48A| D0F7                 bne $4
E48C|
E480* 0A
E48C| 60                 $9  rts             ;  and, done...
E48D|
E48D|
E48D|                   ;*************************************************************************
E48D|                   ;*  wrDMA, write a block of data from area defined by dP/dC to DMAreg.
E48D|                   ;* To increase speed, we do them in chunks of 4, then remainder.
E48D|                   ;*************************************************************************
E48D| A0 00             wrDMA   ldy #0          ; starting offset
E48F| A5 1A                 lda dC          ; pickup low-byte of count
>>>>PAGE -  30  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E491| 66 1B                 ror dC+1            ; shift hi
E493| 6A                   ror A           ;  into lo
E494| 66 1B                 ror dC+1
E496| 6A                   ror A           ;  for 2 bits
E497| F0**                 beq $3          ; skip to remainder case if no 4x
E499| AA                   tax             ;  else, copy for loop's usage
E49A|
E49A| B1 18             $1  lda @dP,y           ; copy 4 fast...
E49C| 8D 0080               sta DMAreg
E49F| C8                   iny
E4A0| B1 18                 lda @dP,y
E4A2| 8D 0080               sta DMAreg
E4A5| C8                   iny
E4A6| B1 18                 lda @dP,y
E4A8| 8D 0080               sta DMAreg
E4AB| C8                   iny
E4AC| B1 18                 lda @dP,y
E4AE| 8D 0080               sta DMAreg
E4B1| C8                   iny
E4B2| D0**                 bne $2          ; we only test after every 4 bytes
E4B4| E6 19                 inc dP+1
E4B2* 02
E4B6| CA                 $2  dex             ; down-count of /4
E4B7| D0E1                 bne $1
E4B9|
E4B9|                   ;** handle remainder 0..3 bytes here
```

```
E497* 20
E4B9| A5 1A             $3  lda dC          ; pickup low 3 bytes
E4BB| 29 03                 and #003           ; use only 2 bits
E4BD| F0**                  beq $9          ; yeah, all done
E4BF| AA                    tax             ;  else, copy for loop
E4C0|
E4C0| B1 18             $4  lda @dP,y           ; last odd ones...
E4C2| 8D 0080               sta DMAreg
E4C5| C8                    iny
E4C6| CA                    dex             ; down count
E4C7| D0F7                  bne $4
E4C9|
E4BD* 0A
E4C9| 60                 $9  rts             ;  and, done...
E4CA|
E4CA|                    ;*************************************************************************
E4CA|
E4CA|                    ;*  RCVtask, this task (which is always "running") is responsible for
E4CA|                    ;* the analysis and processing of any incoming packets.  The low-level LAP
E4CA|                    ;* receiver (rxIntr) will enqueue packets to recvQ, and activate RCVtask
E4CA|                    ;* (via a semaV(RCVtask)).
E4CA|                    ;*************************************************************************
E4CA|                    RCVfree     ;** here for code which has allocated DMA
E4CA| 20 3BE4                jsr freeDMA         ; free up DMA for our task
E4CD|
E4CD|                    RCVtask     ;** all roads (for receive handlers) lead back here!!!!
E4CD| 58                      cli             ; make sure ints are allowed
E4CE| A5 54                   lda RCVBufN         ; is there a buffer still allocated?
E4D0| F0**                    beq $1          ;  nope, skip it
>>>>PAGE -  31  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E4D2| 20 73E2                jsr putFree         ; else, requeue it to freeQ
E4D5| 64 54                  stz RCVBufN
E4D7|
E4D0* 05
E4D7| A5 A5              $1  lda rxP+1           ; is there an active buffer?
E4D9| D0**                    bne RCVwait         ;  yes, we're ok
E4DB| A2 A4                   ldx #rxP            ; no, get one
E4DD| 20 33E2                jsr getFree
E4E0| 20 5BE2                jsr setRXP          ; update our stuff
E4E3|
E4E3|                    ;** the basic loop "wait";  we will be activated by rxIntr...
E4D9* 08
E4E3| A2 A8              RCVwait ldx #RCVsema
E4E5| 20 CCE1                jsr semaP
E4E8|
E4E8|                    ;*  when we get back, at least one packet is sitting in our recvQ
E4E8| 78                      sei         ; inihibit while diddling
E4E9| A6 AC                   ldx recvQhd     ; get front of queue
E4EB|
E4EB| B5 80                   lda freeQ,x     ;  its next
E4ED| 85 AC                   sta recvQhd
E4EF| D0**                    bne $0
E4F1| 85 AD                   sta recvQtl     ; if empty, fix last also
E4F3|
E4F3|                    ;** ok, analyze the packet
E4EF* 02
E4F3| 58                 $0  cli
E4F4| 8A                      txa
E4F5| 85 54                   sta RCVBufN     ; save for later
E4F7| A2 14                   ldx #rtP        ; setup our ptr
E4F9| 20 44E2                jsr cvtBufN
E4FC|
E4FC| A0 04                   ldy #laType+2   ; look at its type
E4FE| B1 14                   lda @rtP,y
E500| C9 01                   cmp #laDDP      ; DDP (short)?
E502| F0**                    beq $10
E504| C9 02                   cmp #laDDPX     ; DDP (extended)?
E506| F0**                    beq $20
E508| 20 ****                 jsr chkLAP      ; check if protocol is in registry
E50B| D0**                    bne $1      ; user wants it
E50D|
E50D| A9 02                   lda #fALAP      ; else, all LAP?
E50F| 24 30                   bit SysFlgs
E511| F0BA                   beq RCVtask     ; if not, forget it
E513|
E50B* 06
E513| 4C ****            $1  jmp xRXLAP      ; else, send it to Host
E516|
E502* 12
E516| A9 04              $10 lda #fNDDP      ; do DDP?
E518| 24 30                   bit SysFlgs
E51A| D0F7                   bne $1      ;  no, handle as LAP
E51C| 4C ****                jmp xRXDDP      ; yes, let someone who knows handle it
E51F|
E506* 17
>>>>PAGE -  32  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk
```

```
E51F| A9 04             $20 lda #fNDDP      ; do DDP?
E521| 24 30                 bit SysFlgs
E523| D0EE                 bne $1      ;  no, handle as LAP
E525| 4C ****               jmp xRXDDPX
E528|
E528|                   ;*************************************************************************
E528|                   ;*  XMTtask, the code to transmit packets.  The packets may either be
E528|                   ;* explicit (via LAP/DDP requests) or implicit (via ATP requests).  In any
E528|                   ;* case, we remove the next request from xmitQ and txPacket it.
E528|                   ;*************************************************************************
E528|
E528| 58                XMTtask cli         ; make sure to allow ints
E529| A2 20                   ldx #XMTsema    ; the xmitQ count (we hope!)
E52B| 20 CCE1               jsr semaP
E52E|
E52E|                   ;** we have a buffer to send, acquire it and send.
E52E| 78                     sei         ; inhibit while checking queue
E52F| A6 24                 ldx xmitQhd     ; get first
E531| F0F5                 beq XMTtask     ; whoops!!!! (shouldn't happen)
E533| B5 80                 lda freeQ,x     ; advance chain
E535| 85 24                 sta xmitQhd
E537| D0**                 bne $1      ; skip if not last
E539| 85 25                 sta xmitQtl     ; else, set last ptr also
E53B|
E53B|                   ;** having obtained a buffer, setup our params
E537* 02
E53B| 58                $1  cli         ; can allow now
E53C| 86 4C                 stx XMTBufN     ; current XMT buffer Number
E53E| A2 78                 ldx #XMTlock    ; obtain user of transmitter
E540| 20 CCE1               jsr semaP
E543|
E543| A2 94                 ldx #txPd
E545| A5 4C                 lda XMTBufN
E547| 20 44E2               jsr cvtBufN     ; convert it
E54A|
E54A|                   ;** convert addresses for header/data
E54A| 18                     clc         ; compute address of header
E54B| A5 94                 lda txPd
E54D| 69 68                 adc #HDROFS % 0100
E54F| 85 90                 sta txPh
E551| A5 95                 lda txPd+1
E553| 69 02                 adc #HDROFS / 0100
E555| 85 91                 sta txPh+1
E557|
E557|                   ;** copy params for txPacket call
E557| B2 94                 lda @txPd       ; size of header
E559| E6 94                 inc txPd
E55B| 85 92                 sta txCh
E55D| 64 93                 stz txCh+1
E55F| B2 94                 lda @txPd       ; requesting task
E561| E6 94                 inc txPd
E563| 48                     pha         ; save it on stack
E564| B2 94                 lda @txPd       ; length of data
E566| E6 94                 inc txPd
E568| 85 96                 sta txCd
>>>>PAGE -  33  PCCARD    FILE: PCCMDS.a65  PCcard - firmware for IBM-PC AppleTalk


E56A| B2 94                 lda @txPd
E56C| E6 94                 inc txPd
E56E| 85 97                 sta txCd+1
E570|
E570|                   ;** finally, send the bloody thing
E570|
E570| 20 ****               jsr txPacket
E573| 48                     pha         ; save result
E574| A2 78                 ldx #XMTlock    ; free tx
E576| 20 98E1               jsr semaV
E579| 68                     pla         ; restore status
E57A| FA                     plx         ;  and task to wakeup
E57B| 95 05                 sta tRslt,x     ; save it for its use
E57D| A9 04                 lda #sTXDONE    ; signal the task
E57F| 20 74E1               jsr Signal
E582| 4C 28E5               jmp XMTtask     ;  and, loop
E585|
E585|                   ;*************************************************************************
E585|                   ;*  xTXPKTA, common code to allocate a buffer and setup its ptrs.
E585|                   ;*  dP points to "data" portion of buffer, t2 pts to "header".
E585|                   ;*************************************************************************
E585|                   xTXPKTA                 ;** used by txLAP/txDDP
E585| A2 18                 ldx #dP         ; get free buffer
E587| 20 33E2               jsr getFree
E58A| F0**                 beq txPKTAe         ; error here
E58C| 85 44                 sta CMDBufN         ; save our buffer number
E58E|
E58E|                   ;** make LAP header in our buffers offset area
```

```
E58E| 18                    clc
E58F| A5 18                 lda dP          ; copy for header build
E591| 69 68                 adc #HDROFS % 0100
E593| 85 02                 sta t2
E595| A5 19                 lda dP+1
E597| 69 02                 adc #HDROFS / 0100
E599| 85 03                 sta t2+1
E59B| 60                    rts
E59C|
E58A* 10
E59C| 68                 txPKTAe pla             ; pop return address
E59D| 68                    pla
E59E| A9 FE                 lda #NoFreeErr      ; no free buffer
E5A0| 4C 45E3              jmp CMDdone
E5A3|
E5A3|                      ;*************************************************************************
E5A3|                      ;**EOF  PCCMDS
E5A3|
E5A3|
E5A3|                         .include    PCLAP       ; Link Access Protocol handler
E5A3|                      ;** PCLAP, PCcard Link Access Protocol code
E5A3|
E5A3|                      ;*************************************************************************
E5A3|                      ;*  RxIntr, entry from interrupt handler upon seeing RCA packet.  Note:
E5A3|                      ;* A is the only reg saved to this point, so finish saving while receiving
E5A3|                      ;* first bytes of frame.
E5A3|                      ;*  In order to allow users to handle non-DDP style (i.e., no explicit
>>>>PAGE -  34  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E5A3|                      ;* length field), we check for EOP on each byte.  We should be running fast
E5A3|                      ;* enough so that we can afford to make this check.  It is the responsibility
E5A3|                      ;* of the higher levels to verify lengths.
E5A3|                      ;*  We assume that the receive params (mainly, rP and rC) have been set
E5A3|                      ;* and we can use them as is;  woe betides us if this is not the case!!!
E5A3|                      ;*************************************************************************
E5A3|
E2C0* A3E5
E5A3| DA                 rxIntr  phx             ; save other regs
E5A4| 5A                    phy
E5A5| AD 0140                lda sccData         ; we know there's one present
E5A8| 48                    pha             ;  save it
E5A9| A5 A1                 lda rP+1            ; check on presence of buffer
E5AB| F0**                  beq rxSKP1          ; whoops, throw it away.
E5AD| 68                    pla             ; else, get the byte
E5AE| A0 02                 ldy #2          ; setup loop (skip over count field)
E5B0|
E5B0| 91 A0                 sta @rP,y           ;  and, save ours
E5B2|
E5B2|                      ;** the main receive loop
E5B2| A2 08                $0  ldx #8          ; timeout value (for UNDerruns)
E5B4| A9 01                    lda #RCA            ; bit loop value
E5B6| C8                    iny             ; up rP offset
E5B7| D0**                  bne $1
E5B9| E6 A1                 inc rP+1
E5B7* 02
E5BB| E6 A2                $1  inc rC          ; adjust max length ctr
E5BD| D0**                  bne $2
E5BF| E6 A3                 inc rC+1
E5C1| 10**                  bpl rxLNG           ; error if too many!
E5C3|
E5BD* 04
E5C3| 2C 0040              $2  bit sccCtrl         ; RCA?
E5C6| D0**                  bne $3          ;  yes, fetch it
E5C8| CA                    dex
E5C9| D0F8                  bne $2          ; loop if not timed out
E5CB| 80**                  bra rxUND           ;  whoops, we stopped!?!
E5CD|
E5C6* 05
E5CD| AD 0140              $3  lda sccData         ; fetch the byte
E5D0| 91 A0                 sta @rP,y           ;  and, save it
E5D2| A9 01                 lda #1          ; check for errors
E5D4| 8D 0040               sta sccCtrl
E5D7| AD 0040               lda sccCtrl
E5DA| 30**                  bmi rxEOF           ; skip on EndOfFrame
E5DC| 29 20                 and #OVR            ; OVeRrun?
E5DE| F0D2                  beq $0          ;  nope, keep going
E5E0|
E5E0|                      ;** error present;  should be EOF
E5E0| A2 BE                rxOVR   ldx #OVRct          ; handle the error
E5E2| 80**                  bra rxReset
E5C1* 21
E5E4| A2 C4                rxLNG   ldx #LNGct
E5E6| 80**                  bra rxReset
E5E8| A2 C0                rxCRC   ldx #CRCct
>>>>PAGE -  35  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk
```

```
E5EA| 80**                  bra rxReset
E5AB* 3F
E5EC| 68                  rxSKP1  pla             ; entry from rxIntr, pop 1st byte
E5ED| A2 BA               rxSKP   ldx #SKPct
E5EF| 80**                  bra rxReset
E5CB* 24
E5F1| A2 BC               rxUND   ldx #UNDct
E5F3| 80**                  bra rxReset
E5F5|
E5DA* 19
E5F5| 29 60               rxEOF   and #CRC+OVR        ;  else, check CRC
E5F7| D0EF                  bne rxCRC           ; if either, treat as CRC
E5F9| A9 30                 lda #030
E5FB| 8D 0040               sta sccCtrl         ; reset Errors
E5FE| AD 0140               lda sccData         ;  and, flush the FIFO
E601| A0 04                 ldy #laType+2       ; look at control byte
E603|
E603| B1 A4                  lda @rxP,y
E605| 30**                   bmi rxLAP           ; LAP control (maybe!?)
E607| 4C ****                jmp rxData          ; its a Data frame, handle it
E60A|
E60A|                     ;** the byte looks like a control packet, check on type
E605* 03
E60A| 85 AE               rxLAP   sta fType           ; save what we got
E60C| C9 84                 cmp #laRTS          ; RequestToSend?
E60E| D0**                  bne $1
E610| 4C ****               jmp rxRTS           ;  yep, reply (unless broadcast)
E613|
E60E* 03
E613| C9 81               $1  cmp #laENQ          ; only other "valid" one
E615| D0**                  bne $2
E617| 4C ****               jmp rxENQ           ; ENQUIRY, defend our number
E61A|
E61A|                     ;** others are either illegal, or duplicate number
E615* 03
E61A| C9 85               $2  cmp #laCTS          ; CTS (to us?!)
E61C| F0**                  beq rxCTS
E61E| C9 82                 cmp #laACK
E620| D0**                  bne rxBad
E622|                     rxACK
E61C* 04
E622|                     rxCTS
E622| A2 B8                  ldx #CTSct          ; set for normal exit
E624| A5 AF                  lda laState         ; were we doing Tx?
E626| 30**                   bmi rxReset         ; yep, count and leave
E628|
E628| A2 C6               $1  ldx #DUPct
E62A| 80**                  bra rxReset
E62C|
E620* 0A
E62C| A2 C2               rxBad   ldx #BADct          ; update error counter
E62E|
E62E|                     ;*************************************************************************
E62E|                     ;*  rxReset, reset Rx after valid packet, or in middle of bad one.  X is
E62E|                     ;* assumed to have the base of an error counter to be adjusted.
>>>>PAGE -  36  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E62E|                     ;*************************************************************************
E62A* 02
E626* 06
E5F3* 39
E5EF* 3D
E5EA* 42
E5E6* 46
E5E2* 4A
E62E| F6 00               rxReset inc 0,x         ; update low
E630| D0**                  bne $1
E632| E8                    inx
E633| F6 00                 inc 0,x
E635|
E635|                     ;** complete by resetting working regs (rP,rC) to correct values
E635|
E630* 03
E635| A5 A4               $1  lda rxP         ; copy to working set
E637| 85 A0                  sta rP
E639| A5 A5                  lda rxP+1
E63B| 85 A1                  sta rP+1
E63D| A9 9F                  lda #~<rxCsz % 0100>    ; set buffer size (compl)
E63F| 85 A2                  sta rC
E641| A9 FD                  lda #~<rxCsz / 0100>
E643| 85 A3                  sta rC+1
E645| 20 ****                jsr rxFlush         ; flush Rx, (and, fall thru to rxRTI)
E648|
E648|                     ;*************************************************************************
E648|                     ;*  RxRTI, return from Rx interrupt;  off to common post-processing.
E648|                     ;*************************************************************************
E648| E6 B0               rxRTI   inc INTct           ; update count of interrupt
```

```
E64A| D0**                bne $1
E64C| E6 B1               inc INTct+1
E64A* 02
E64E| 7A                  $1  ply
E64F| FA                      plx
E650| 4C ADE2                 jmp SCCintX         ;  and, look for others
E653|
E653|                     ;*************************************************************************
E653|                     ;*  rxFlush, reset SCC, flushing any residual bytes in FIFO.  Note, this
E653|                     ;* code gets called by both Rx and Tx code.
E653|                     ;*************************************************************************
E646* 53E6
E653|                     rxFlush
E653| A2 03                     ldx #3.         ; reset RX, inline to be fast..
E655| 8E 0040                   stx sccCtrl
E658| A9 C0                     lda #0C0
E65A| 8D 0040                   sta sccCtrl
E65D|
E65D| AD 0040              $1   lda sccCtrl         ; still something/
E660| 6A                       ror A           ; note!! trick here, C gets RCA
E661| 90**                     bcc $2          ;  nope, continue
E663| A9 30                     lda #030            ; else, reset Errors
E665|
E665| 8D 0040                   sta sccCtrl
E668| AD 0140                   lda sccData         ;  and, pop the FIFO
>>>>PAGE -  37  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E66B| 80F0                     bra $1
E66D|
E661* 0A
E66D| 8E 0040              $2   stx sccCtrl
E670| A9 DD                     lda #0DD
E672| 8D 0040                   sta sccCtrl
E675| A9 20                     lda #020            ; reset for Rx intr
E677| 8D 0040                   sta sccCtrl
E67A| A9 30                     lda #030            ; final (extra) reset Errors
E67C| 8D 0040                   sta sccCtrl
E67F| 60                       rts
E680|
E680|                     ;*************************************************************************
E680|                     ;*  rxRTS, a RequestToSend has been seen.  Send a CTS (unless a broadcast)
E680|                     ;* and wait for assumed data packet.
E680|                     ;*  rxENQ, likewise, respond w/ ACK to ENQ.
E680|                     ;*************************************************************************
E680|
E618* 80E6
E680| A9 82              rxENQ   lda #laACK          ; respond w/ ACK
E682| D0**                       bne rxXXX
E684|
E611* 84E6
E684| A9 01              rxRTS   lda #laDATwt
E686| 85 AF                     sta laState         ; show what's happening
E688| A9 85                     lda #laCTS          ; set to recv, send a CTS
E68A|
E68A|                     ;** turn packet around
E682* 06
E68A| 85 6E              rxXXX   sta CTSfrm+laType       ; stash our response type
E68C| A5 13                     lda myFlags         ; have we determined our number yet?
E68E| 30**                     bmi $1          ; yes, its ok..
E690| 4C EDE5                   jmp rxSKP           ;  no, ignore it
E693|
E68E* 03
E693| A0 02              $1   ldy #laDst+2        ; swap Dst/Src
E695| B1 A4                   lda @rxP,y
E697| C9 FF                   cmp #0FF            ; was this a broadcast?
E699| F0**                   beq $9          ;  yes, skip CTS send
E69B|
E69B|                     ;** not a broadcast, setup la stuff in CTSfrm for txFrame response
E69B| C8
E69C| B1 A4                   lda @rxP,y          ; fetch Src
E69E| 85 6C                   sta CTSfrm+laDst
E6A0| A9 6C                   lda #CTSfrm         ; setup txFrame args
E6A2| 85 D0                   sta xP
E6A4| A9 03                   lda #3
E6A6| 85 D2                   sta xC
E6A8| 64 D1                   stz xP+1
E6AA| 64 D3                   stz xC+1
E6AC| 20 ****                 jsr txFrame         ; send the sucker
E6AF|
E699* 14
E6AF| A2 B6              $9   ldx #RTSct          ; show we went ok
E6B1| 4C 2EE6                 jmp rxReset
>>>>PAGE -  38  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E6B4|
E6B4|                     ;*************************************************************************
```

```
E6B4|                   ;*  rxData, received a data frame.  Enqueue the packet to recvQ, and
E6B4|                   ;* activate RCVtask via its semaphore.  Note that we load rC w/ complement
E6B4|                   ;* of rxCsz.  This value is incremented for each byte PLUS 1 additional for
E6B4|                   ;* the 1st CRC byte.  By adding rxCsz, we adjust for this one extra!
E6B4|                   ;*************************************************************************
E608* B4E6
E6B4| A0 00             rxData  ldy #0          ; prepare to stash size
E6B6| 18                    clc             ; compute size of packet
E6B7| A5 A2                 lda rC
E6B9| 69 60                 adc #rxCsz % 0100
E6BB| 91 A4                 sta @rxP,y          ;  and, place at front of buffer
E6BD| C8                    iny
E6BE| A5 A3                 lda rC+1
E6C0| 69 02                 adc #rxCsz / 0100
E6C2| 91 A4                 sta @rxP,y
E6C4|
E6C4| A5 A6                 lda rxBufN          ; our buffer's number
E6C6| A6 AD                 ldx recvQtl         ; add it to end
E6C8| 85 AD                 sta recvQtl         ; our's is new end
E6CA| D0**                  bne $1
E6CC| 85 AC                 sta recvQhd         ; if empty, place ours on head
E6CE| 80**                  bra $2
E6D0|
E6CA* 04
E6D0| 95 80             $1  sta freeQ,x         ; last's next
E6D2|
E6CE* 02
E6D2| AA                $2  tax
E6D3| 74 80                 stz freeQ,x         ; make sure or our next
E6D5| A2 A8                 ldx #RCVsema        ; then, activate RCVtask
E6D7| 20 98E1               jsr semaV
E6DA|
E6DA| A2 A4                 ldx #rxP            ; get another buffer (if available)
E6DC| 20 33E2               jsr getFree
E6DF| 20 5BE2               jsr setRXP          ; and, setup stuff
E6E2|
E6E2| A2 B2                 ldx #RXPct          ; if we got this far, we're home
E6E4| 4C 2EE6               jmp rxReset
E6E7|
E6E7|                   ;*************************************************************************
E6E7|                   ;*  xRXLAP, handler for non-DDP packets (filtered by RCVtask).
E6E7|                   ;*************************************************************************
E514* E7E6
E6E7| 20 ****         xRXLAP  jsr xRXPKT1         ; obtain DMA
E6EA| A9 14                 lda #014            ; rxLAP "function"
E6EC|
E6EC| 8D 0080               sta DMAreg          ; function code
E6EF| A2 03                 ldx #3          ; length passed during DMAinfo phase
E6F1| A0 00                 ldy #0          ; copy length for computation
E6F3| 38                    sec
E6F4| B1 14                 lda @rtP,y
E6F6| C8                    iny
E6F7| E9 03                 sbc #3
>>>>PAGE -  39  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E6F9| 85 1A                 sta dC          ; set remainder data
E6FB| B1 14                 lda @rtP,y
E6FD| C8                    iny
E6FE| E9 00                 sbc #0
E700| 85 1B                 sta dC+1
E702|
E702| B1 14             $1  lda @rtP,y          ; stuff it
E704| C8                    iny
E705| 8D 0080               sta DMAreg
E708| CA                    dex
E709| D0F7                 bne $1
E70B| A5 1A                 lda dC          ; stash to user (length)
E70D| 8D 0080               sta DMAreg
E710| A5 1B                 lda dC+1
E712| 8D 0080               sta DMAreg
E715|
E715| 98                    tya             ; setup dP
E716| 18                    clc
E717| 65 14                 adc rtP
E719| 85 18                 sta dP
E71B| A5 15                 lda rtP+1
E71D| 85 19                 sta dP+1
E71F|
E71F|                   ;*************************************************************************
E71F|                   ;*  xRXPKT2, common tail-end processing for received packet.
E71F|
E71F|                   ;*************************************************************************
E71F|                   xRXPKT2
E71F| A9 D0                   lda #DMAdata        ; show ready for data now
E721| 20 24E4               jsr waitDMA
E724|
E724|                   ;** now, copy the packet (note: dC has already been setup by "caller")
```

```
E724| 20 8DE4              jsr wrDMA           ; and, do it
E727|
E727| 4C CAE4              jmp RCVfree         ;  and, we're ready for more
E72A|
E72A|                    ;*************************************************************************
E72A|                    ;*  xRXPKT1, front-end common code for receiving a packet.
E72A|                    ;*************************************************************************
E6E8* 2AE7
E72A| A2 74              xRXPKT1 ldx #DMAlock        ; for valid LAP pkt, grab DMA channel
E72C| 20 CCE1              jsr semaP
E72F| A9 C0                lda #DMAinfo        ; tell user what we want
E731| 20 24E4              jsr waitDMA
E734| 60                  rts
E735|
E735|                    ;*************************************************************************
E735|                    ;*  xINITLAP, initialization code for LAP (and card).  Until the first
E735|                    ;* INITLAP, the Rx is disabled.  Here, we setup the SCC for our port, and
E735|                    ;* start sending ENQ frames, looking for anyone responding.
E735|                    ;*************************************************************************
E735|                    xINITLAP
E735| AD 0080              lda DMAreg          ; user's guess at a number
E738|
E738| 85 69                sta myNode          ; stash it
>>>>PAGE -  40  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E73A| A9 80                lda #fMyNode        ; make sure we don't look valid now
E73C| 14 13                trb myFlags
E73E| A2 78                ldx #XMTlock        ; grab Tx for a while
E740| 20 CCE1              jsr semaP
E743|
E743|                    ;** initialize xSeed from SCC counter in A-port
E743| A5 DE                lda xSeed
E745| 05 DF                ora xSeed+1         ; set yet?
E747| D0**                 bne $0          ;  yep, we only do this first time...
E749| A9 0D                lda #13.
E74B| 78                   sei
E74C| 8D 0240              sta SCCctrlA
E74F| AD 0240              lda SCCctrlA
E752| 58                   cli
E753| 85 DE                sta xSeed
E755| A9 0C                lda #12.
E757| 78                   sei
E758| 8D 0240              sta SCCctrlA
E75B| AD 0240              lda SCCctrlA
E75E| 58                   cli
E75F| 85 DF                sta xSeed+1
E761|
E761|                    ;** the retry loop, if an attempt fails (or, first time)
E747* 18
E761| A9 81              $0  lda #laENQ          ; set RTS frame for ENQ
E763| 85 6A                sta RTSfrm+laType
E765| A5 69                lda myNode          ; get users again
E767| D0**                 bne $2          ;  if non-zero, try it first
E769|
E769| 26 69              $1  rol myNode          ; save upper bit
E76B| 64 69                stz myNode
E76D| 66 69                ror myNode
E76F| 20 ****              jsr Random          ; generate a value
E772| 29 7F                and #07F            ; mask to dynamic area
E774| 05 69                ora myNode          ; add in user/server bit
E776| F0F1                beq $1          ;  try again if 0
E778|
E778|                    ;** initialize our RTS pkt
E767* 0F
E778| 85 69              $2  sta myNode          ; save for a while (aka, RTSfrm+laSrc)
E77A| 85 68                sta RTSfrm+laDst        ; (we send to ourselves)
E77C| 85 6D                sta CTSfrm+laSrc        ;  save for our responses!!
E77E| A2 06                ldx #6.
E780| 78                   sei
E781| 8E 0040              stx SCCctrl         ; setup SCC for this try
E784| 8D 0040              sta SCCctrl
E787| 58                   cli
E788|
E788| A9 05                lda #5.         ; a retry counter
E78A|
E78A| 85 03                sta t3
E78C| A9 68                lda #RTSfrm         ; fake a frame
E78E| 85 90                sta txPh
E790| 64 91                stz txPh+1          ; (prevents data Tx at txPkt5)
E792|
>>>>PAGE -  41  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E792|                    ;** the testing loop here.
E792| 20 ****            $3  jsr txPacket        ; try to send it again
E795| F0D2                beq $1          ; in use, must pick another
E797| C6 03                dec t3          ; ok, try again?
```

```
E799| 30**                 bmi $9          ; no, we probably have it!!
E79B|
E79B|                   ;** sync up to next second boundary
E79B| A5 1C              lda TMRsecs         ; get current time
E79D| C5 1C              $4  cmp TMRsecs         ;  and, wait for change
E79F| F0FC                  beq $4
E7A1| 80EF                  bra $3          ; then, try another time
E7A3|
E7A3|                   ;** we've tried a few times, and no one complained...
E799* 08
E7A3| A9 80              $9  lda #fMyNode        ; show we're now ready
E7A5| 04 13                 tsb myFlags
E7A7| A9 84                 lda #laRTS          ; fixup work frame
E7A9| 85 6A                 sta RTSfrm+laType
E7AB| A2 78                 ldx #XMTlock        ; free up Tx
E7AD| 20 98E1               jsr semaV
E7B0|
E7B0| 4C ****               jmp setRTMP         ; exit thru RTMP module     ;-A
E7B3|
E7B3|                   ;*************************************************************************
E7B3|                   ;*  xADDLAP, add a LAP type to our receive LAPtbl.
E7B3|                   ;*************************************************************************
E7B3| AD 0080            xADDLAP lda DMAreg          ; get LAP type
E7B6| 30**                 bmi ADDDELe         ;  error!!
E7B8|
E7B8|                   ;** use code to address table, set mask.
E7B8| 20 ****               jsr chkLAP          ; leaves X w/ offset, A w/ mask
E7BB| D0**                 bne ADDDELw         ;  error if so
E7BD|
E7BD| 1D 6001               ora LAPtbl,x        ; else, add it in
E7C0|
E7C0| 9D 6001            ADDDELx sta LAPtbl,x        ; update table's byte
E7C3| A9 00                 lda #0          ; no error exit
E7C5| 4C 45E3               jmp CMDdone
E7C8|
E7BB* 0B
E7C8| A9 01            ADDDELw lda #SKTerr         ; warning only
E7CA| 4C 45E3               jmp CMDdone
E7B6* 15
E7CD| A9 FF            ADDDELe lda #REQerr         ; error if <0 protocol
E7CF| 4C 45E3               jmp CMDdone
E7D2|
E7D2|                   ;*************************************************************************
E7D2|                   ;*  xDELLAP, delete a LAP type from our LAPtbl.
E7D2|                   ;*************************************************************************
E7D2| AD 0080            xDELLAP lda DMAreg          ; fetch user's param
E7D5| 30F6                 bmi ADDDELe         ;  obvious problem
E7D7|
E7D7| 20 ****               jsr chkLAP          ; is it present?
E7DA| F0F1                  beq ADDDELe         ;  nope, error
E7DC| 5D 6001               eor LAPtbl,x        ; yes, remove it
>>>>PAGE -  42  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E7DF| 80DF                  bra ADDDELx         ;  and, share code
E7E1|
E7E1|                   ;*************************************************************************
E7E1|                   ;*  chkLAP, lookup protocol (in A), does BIT here, leaves Z set.
E7E1|                   ;*************************************************************************
E7D8* E1E7
E7B9* E1E7
E509* E1E7
E7E1| A8                chkLAP  tay             ; save protocol
E7E2| 4A                    lsr A           ; shift off offset
E7E3| 4A                    lsr A
E7E4| 4A                    lsr A
E7E5| AA                    tax             ; X is offset into LAPtbl
E7E6| 98                    tya             ; get protocol back
E7E7| 29 07                 and #007            ; bit#
E7E9| A8                    tay
E7EA| B9 4BE0               lda BITS,y          ; get bit mask
E7ED|
E7ED| 3C 6001               bit LAPtbl,x        ; test current bit
E7F0| 60                    rts
E7F1|
E7F1|                   ;*************************************************************************
E7F1|                   ;*  xTXLAP, processing code for the request to send a packet from user.
E7F1|                   ;* Note that this is a two-phase command and does not actually initiate
E7F1|                   ;* packet transmission directly.  XMTtask will actually perform the dirty
E7F1|                   ;* work.  All we do here is setup a buffer, copy user's data and wait for
E7F1|                   ;* XMTtask to Signal us when the packet has been sent.
E7F1|                   ;*************************************************************************
E7F1|                   xTXLAP                  ;** entry from CMDtask
E7F1| 20 85E5               jsr xTXPKTA         ; allocate buffer
E7F4| A0 00                 ldy #0
E7F6| AD 0080               lda DMAreg          ; get dstNode
E7F9| 91 02                 sta @t2,y
E7FB| C8                    iny
```

```
E7FC| A5 69               lda myNode          ; source # (me!!)
E7FE| 91 02               sta @t2,y
E800| C8                  iny
E801| AD 0080             lda DMAreg          ; laType
E804| 91 02               sta @t2,y
E806| AD 0080             lda DMAreg          ; pktsize(lo)
E809| 85 1A               sta dC
E80B| AD 0080             lda DMAreg          ; pktsize(hi)
E80E| 85 1B               sta dC+1
E810| A9 03               lda #3          ; length of LAP header
E812|
E812|                   ;*************************************************************************
E812|                   ;*  xTXPKTX, common code shared by LAP/DDP.
E812|                   ;*  xTXPKTZ, common entry to clean up after a send.
E812|                   ;*************************************************************************
E812|
E812| A2 40             xTXPKTX ldx #CMDtcb         ; set param for xTXPKTD
E814| 20 ****               jsr xTXPKTD         ; send the packet
E817| A5 44               lda CMDBufN
E819| 64 40               stz CMDFlgs         ; clear other flags
E81B| 20 80E2             jsr putXmit
>>>>PAGE -  43  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E81E|
E81E| A9 04               lda #sTXDONE        ; show what we're waiting on
E820| 20 64E1             jsr Wait
E823|
E823| A5 44             xTXPKTZ lda CMDBufN         ; get our buffer number back
E825| 20 73E2             jsr putFree         ; add it back to free list
E828| 4C 4CE3             jmp CMDstat         ;  and, we're done...
E82B|
E82B|                   ;*************************************************************************
E82B|                   ;*  xTXPKTD, common code to acquire "data" portion of packet, Note that
E82B|                   ;* this code is shared by xTXLAP/xTXDDP/xTXDDPX/xNBLKUP.
E82B|                   ;*  dP/dC must be setup;  it is assumed that dC has proper length and
E82B|                   ;* that dP points to buffer.  We setup 1st 4 bytes to pass stuff to XMTtask.
E82B|                   ;* A has length of the header (HDROFS) for this buffer; X has TCB ptr.
E82B|                   ;*************************************************************************
E815* 2BE8
E82B|                   xTXPKTD                 ;* jsr'd by above or pcDDP
E82B|                   ;** build front of buffer w/ info req'd by XMTtask
E82B| A0 00               ldy #0          ; make sure of index
E82D| 91 18               sta @dP,y
E82F| C8                  iny
E830| 8A                  txa             ; task for Signal by XMTtask
E831| 91 18               sta @dP,y
E833| C8                  iny
E834| A5 1A               lda dC          ; copy counter
E836|
E836| 91 18               sta @dP,y
E838| C8                  iny
E839| A5 1B               lda dC+1
E83B| 91 18               sta @dP,y
E83D| 18                  clc
E83E| A9 04               lda #4          ; adjust dP for rdDMA
E840| 65 18               adc dP
E842| 85 18               sta dP
E844|
E844|                   ;** now, request and copy user's data for this packet.
E844| A9 B1               lda #REQdata+1      ; show why
E846| 20 24E4             jsr waitDMA         ;  and, wait for it
E849| 20 50E4             jsr rdDMA           ; ok, read packet's data
E84C| A5 11               lda pcIdle          ; free up DMA
E84E| 20 80E0             jsr setSTAT         ;  for Host
E851| A2 74               ldx #DMAlock        ; we're done w/ DMA for now
E853| 20 98E1             jsr semaV
E856|
E856| 60                  rts
E857|
E857|                   ;*************************************************************************
E857|                   ;*  txPacket, hi-level packet transmission.  We here from TaskLP, or
E857|                   ;* possibly from higher-level protocol handlers, by which time the global
E857|                   ;* parameters (txP/txC) have been setup.
E857|                   ;*************************************************************************
E793* 57E8
E857|                   txPacket    ;** the following is performed only once per transmission
E857| A5 D4               lda xChist          ; adjust backoff
E859| 20 ****             jsr bitCount
>>>>PAGE -  44  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E85C| C9 03               cmp #3.
E85E| 90**                bcc $1
E860| A5 D8               lda xGmask          ; Collsn > 2, increase global mask
E862| 2A                  rol A           ;  C set by cmp
E863| 29 0F               and #00F            ; masked to 15
E865| 85 D8               sta xGmask
```

```
E867| 64 D4                stz xChist          ; clear Collsn history
E869| 80**                 bra $2          ; setup rest of stuff
E86B|
E85E* 0B
E86B| A5 D6             $1  lda xDhist          ; check Defer histories
E86D|
E86D| 20 ****              jsr bitCount
E870| C9 02                cmp #2.
E872| B0**                 bcs $2          ; no adjustment
E874| A5 D8                lda xGmask          ; Defers < 2, decrease mask
E876| 4A                   lsr A
E877| 85 D8                sta xGmask
E879| A9 FF                lda #0FF
E87B| 85 D6                sta xDhist          ; Defers to max
E87D|
E872* 09
E869* 12
E87D| 06 D4             $2  asl xChist          ; shift histories
E87F| 06 D6                 asl xDhist
E881| A5 D8                lda xGmask          ; Global mask -> local
E883| 85 D9                sta xLmask
E885| A9 20                lda #32.            ; setup global retry stuff
E887| 85 D5                sta xCtrys
E889| 85 D7                sta xDtrys
E88B|
E88B| 64 DB                stz xfBcast         ; reset broadcast bit
E88D| B2 90                lda @txPh           ; get laDst of user's
E88F| 85 68                sta RTSfrm+laDst        ;  and, plug to our packet
E891| C9 FF                cmp #0FF            ; broadcast?
E893| D0**                 bne txPkt0
E895| 85 DB                sta xfBcast         ; yes, set the flag (sign!!)
E897|
E897|                    ;*************************************************************************
E897|                    ;*  The following loop is the main transmit code for an entire packet.  By
E897|                    ;* this time, all adjustments to global mask have been made.  We take it from
E897|                    ;* there.
E897|                    ;*************************************************************************
E893* 02
E897| 20 ****           txPkt0  jsr Random          ; gen next random value
E89A|
E89A|                    ;** check if its currently busy
E89A| AD 0040              lda sccCtrl
E89D| 29 10                and #HUNT
E89F| D0**                 bne txPkt1          ; it is idle, start big wait
E8A1|
E8A1|                    ;** line looks busy, time it in case of "stuck" HUNT
E8A1|
E8A1| A5 D9           txIdlWt lda xLmask          ; do min 0..1 delay
E8A3| 09 01                ora #001
>>>>PAGE -  45  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E8A5| 85 D9                sta xLmask
E8A7|
E8A7| A2 0F                ldx #15.
E8A9| A0 00                ldy #0
E8AB| AD 0040           $1  lda sccCtrl
E8AE| 29 10                and #HUNT
E8B0| D0**                 bne txPkt1          ;  ok, it just went idle
E8B2| 88                   dey
E8B3| D0F6                bne $1
E8B5| CA                   dex
E8B6| D0F3                bne $1
E8B8|
E8B8| 20 53E6              jsr rxFlush         ; clean out Rx
E8BB| 4C 97E8              jmp txPkt0          ;  should make it this time
E8BE|
E8BE|                    ;** line looks idle, so wait required IDG time
E8B0* 0C
E89F* 1D
E8BE| 64 AE           txPkt1  stz fType
E8C0| A2 01                ldx #1.         ; first of min
E8C2| 20 ****              jsr ckIdle
E8C5| B0DA                bcs txIdlWt         ; if not idle, defer
E8C7| A2 0E                ldx #14.            ; reset missing clocks
E8C9| A9 41                lda #041
E8CB| 78                   sei
E8CC| 8E 0040              stx sccCtrl
E8CF| 8D 0040              sta sccCtrl
E8D2| 58                   cli
E8D3| A2 03                ldx #3.         ; reset of min
E8D5| 20 ****              jsr ckIdle
E8D8| B0C7                bcs txIdlWt
E8DA|
E8DA|                    ;** now wait the random spread
E8DA| A5 DE                lda xSeed           ; get latest random value
E8DC| 25 D9                and xLmask          ; masked by delay
E8DE| F0**                 beq txPkt2          ; if none, start Tx now!!
```

```
E8E0| AA                    tax             ;  else, copy as param for ckIdle
E8E1|
E8E1| 20 ****               jsr ckIdle          ; delay the extra
E8E4| 90**                  bcc txPkt2          ; yippy, it looks free
E8E6|
E8E6|                   ;** txDefer, perform deference processing here
E8E6| C6 D7              txDefer dec xDtrys          ; more to go before giving up?
E8E8| 10**                   bpl $1          ;  yep
E8EA| A9 EF                  lda #DeferErr       ; nope, report failure
E8EC| 60                    rts
E8ED|
E8E8* 03
E8ED| A5 D6              $1  lda xDhist          ; update history data
E8EF| 09 01                  ora #001
E8F1| 85 D6                  sta xDhist
E8F3| 4C 97E8                jmp txPkt0          ;  and, try again
E8F6|
E8F6|
E8F6|                   ;** we have waited min time, check for other guy possibly starting
E8E4* 10
>>>>PAGE -  46  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E8DE* 16
E8F6| A9 0A              txPkt2  lda #10.
E8F8| 78                    sei
E8F9| 8D 0040                sta sccCtrl
E8FC| AD 0040                lda sccCtrl
E8FF| 58                    cli
E900| 29 C0                  and #MCmask         ; missing clocks?
E902| D0E2                  bne txDefer         ;  yep, defer..
E904|
E904|                   ;** ok, it looks like we have line, so go for it
E904| 78                  txPkt3  sei             ; disable ints here
E905| A2 05                  ldx #5.         ; enable 422, not Tx
E907| 8E 0040                stx sccCtrl
E90A| A9 E2                  lda #0E2
E90C| 8D 0040                sta sccCtrl
E90F| 64 D1                  stz xP+1            ; kill some time, usefully
E911| 64 D3                  stz xC+1
E913| A9 E0                  lda #0E0            ; then, disable 422
E915| 8E 0040                stx sccCtrl
E918| 8D 0040                sta sccCtrl
E91B| A9 03                  lda #3          ; finish up params (and MC delay)
E91D| 85 D2                  sta xC
E91F| A9 68                  lda #RTSfrm
E921| 85 D0                  sta xP
E923| 20 ****                jsr txFrame         ; finally, our frame
E926| 58                    cli             ;  can allow ints now
E927|
E927|                   ;** now, wait for the CTS
E927| A9 FF              txPkt4  lda #laCTSwt        ; show what state
E929| 85 AF                  sta laState         ;  we're in
E92B|
E92B| A2 02                  ldx #2          ; wait 200 usec.
E92D| A0 06                  ldy #6.         ; adjust for end of txFrame
E92F| 20 ****                jsr ckIdle1
E932| B0**                  bcs $1          ; if something in, check it
E934| A5 DB                  lda xfBcast         ; broadcast?
E936| 30**                  bmi txPkt5          ;  yes, send the laDATA
E938| 80**                  bra txColsn         ; else, treat as collison
E93A|
E93A|                   ;** saw line active; wait a slight time longer in case its to us
E932* 06
E93A| A5 DB              $1  lda xfBcast         ; are we broadcasting?
E93C| 30**                  bmi txColsn         ; yes, treat as collision
E93E|
E93E|
E93E| A2 00                  ldx #0          ; else, wait some more
E940| A5 AE              $3  lda fType
E942| D0**                  bne $4
E944| CA                    dex
E945| D0F9                  bne $3
E947| F0**                  beq txColsn         ; take collision exit
E949|
E942* 05
E949| C9 85              $4  cmp #laCTS          ; ours?
>>>>PAGE -  47  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E94B| F0**                  beq txPkt5          ;  yes, send the data pkt
E94D| C9 82                  cmp #laACK          ; ACK?
E94F| F0**                  beq txPktX          ;  yes, treat as normal
E951|
E951|                   ;** txColsn, perform collision processing
E947* 08
E93C* 13
E938* 17
```

```
E951| C6 D5             txColsn dec xCtrys          ; more tries left?
E953| 10**                  bpl $1          ;  yep
E955| A9 EE                 lda #ColsnErr       ; nope, report failure
E957| 60                   rts
E958|
E953* 03
E958| A5 D4             $1  lda xChist          ; update history data
E95A| 09 01                ora #001
E95C| 85 D4                sta xChist
E95E| A5 D9                lda xLmask          ; increase local mask
E960| 38                   sec
E961| 2A                   rol A
E962| 29 0F                and #00F            ; upto 15
E964| 85 D9                sta xLmask
E966| 4C 97E8              jmp txPkt0          ;  and, retry
E969|
E969|                   ;** send user's packet here
E94B* 1C
E936* 31
E969| 78                txPkt5  sei             ; we can't be bothered here
E96A| A5 90                lda txPh            ; setup txFHDR
E96C| 85 D0                sta xP
E96E| A5 91                lda txPh+1
E970| F0**                beq $9          ; special case for xINITLAP
E972| 85 D1                sta xP+1
E974| A5 92                lda TxCh
E976| 85 D2                sta xC
E978| A5 93                lda TxCh+1
E97A| 85 D3                sta xC+1
E97C| 20 ****              jsr txFHDR          ; ok, do first part
E97F|
E97F|                   ;*****************************************************************************
E97F|                   ;*  to save time, and since this is the only routine which sends a
E97F|                   ;* packet, we do the second part here.  Note that the txPd/txCd are not
E97F|                   ;* available after this, but that's OK.
E97F|
E97F|                   ;*****************************************************************************
E97F| A0 00                ldy #0          ; offset counter
E981| A6 96                ldx txCd            ; setup loop
E983| D0**                bne $1
E985| C6 97                dec txCd+1
E987| 30**                bmi $8          ; if none, skip it
E989|
E983* 04
E989| A9 04             $1  lda #TBE            ; the secondary loop
E98B| 2C 0040           $2  bit sccCtrl
E98E| F0FB                beq $2
>>>>PAGE -  48  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E990| B1 94                lda @txPd,y
E992| 8D 0140              sta sccData
E995| CA                   dex
E996| D0**                bne $3
E998| C6 97                dec txCd+1
E99A| 30**                bmi $8          ; loop exit here..
E996* 04
E99C| C8                ;3  iny             ; up offset
E99D| D0EA                bne $1
E99F| E6 95                inc txPd+1
E9A1| 80E6                bra $1
E9A3|
E99A* 07
E987* 1A
E9A3| 20 ****          $8  jsr txFFCS          ; finish off the data frame
E970* 34
E9A6| 58              $9  cli             ; we can allow ints again
E9A7| E6 B4                inc TXPct           ;  and, up the counter
E9A9| D0**                bne txPktX
E9AB| E6 B5                inc TXPct+1
E9AD|
E9A9* 02
E94F* 5C
E9AD| A9 00           txPktX  lda #0          ; show no error
E9AF| 60                   rts
E9B0|
E9B0|                   ;*****************************************************************************
E9B0|                   ;*  txFrame, code to transmit a single frame.  The pointer to the frame
E9B0|                   ;* is in xP, the count is in xC.  In order to share code, txFrame in turn
E9B0|                   ;* uses txFHDR to send front, and falls thru to txFFCS to end it.
E9B0|                   ;*  txPacket uses them also...
E9B0|                   ;*****************************************************************************
E924* B0E9
E6AD* B0E9
E9B0| 20 ****          txFrame jsr txFHDR          ; send header (no secondary)
E9B3|
E9B3|                   ;*****************************************************************************
E9B3|
```

```
E9B3|                   ;*  txFFCS, transmit end of frame stuff here
E9B3|                   ;*****************************************************************************
E9A4* B3E9
E9B3|                   txFFCS
E9B3| AD 0040           $1  lda sccCtrl         ; wait for UND
E9B6| 29 40                 and #EOM
E9B8| F0F9                 beq $1
E9BA|
E9BA|                   ;** then, for TBE (-> FLAG is going out)
E9BA| AD 0040           $2  lda sccCtrl
E9BD| 29 04                 and #TBE
E9BF| F0F9                 beq $2
E9C1|
E9C1|                   ;** disable Tx, (but FLAG will continue)
E9C1| A2 05                 ldx #5.
E9C3| 8E 0040               stx sccCtrl
E9C6| A9 E2                 lda #0E2            ; disables Tx, but leaves 422 on
>>>>PAGE -  49  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


E9C8| 8D 0040               sta sccCtrl
E9CB|
E9CB|                   ;** delay for the abort sequence
E9CB| A0 25                 ldy #37.            ; s.b. about 16 bit times?
E9CD| 88                 $3  dey
E9CE| D0FD                 bne $3
E9D0| 8E 0040               stx sccCtrl
E9D3| A9 E0                 lda #0E0
E9D5| 8D 0040               sta sccCtrl         ; then, 422 off also
E9D8| A9 03                 lda #3.
E9DA| 8D 0040               sta sccCtrl         ; re-arm Rx fast
E9DD| A9 DD                 lda #0DD
E9DF| 8D 0040               sta sccCtrl
E9E2|
E9E2| A9 0E                 lda #14.            ; reset missing clocks
E9E4| 8D 0040               sta sccCtrl
E9E7| A9 41                 lda #041
E9E9| 8D 0040               sta sccCtrl
E9EC| 64 AE                 stz fType
E9EE|
E9EE| 60                   rts             ; finally, return.
E9EF|
E9EF|                   ;*****************************************************************************
E9EF|                   ;*  txFHDR, common routine (used by txFrame and txPacket) to initiate
E9EF|                   ;* transmission of a frame.  This code uses the xP/xC params, which are
E9EF|                   ;* shared by both rxIntr and txPacket.
E9EF|                   ;*****************************************************************************
E9EF|
E9B1* EFE9
E97D* EFE9
E9EF| A9 05             txFHDR  lda #5.         ; start sending FLAGs
E9F1| 8D 0040               sta sccCtrl
E9F4| A9 EB                 lda #0EB
E9F6| 8D 0040               sta sccCtrl         ; enable Tx
E9F9| A2 20                 ldx #rxRst
E9FB| A0 02                 ldy #2
E9FD| 20 64E0               jsr xSCCB           ; disable Rx
EA00|
EA00| A0 05                 ldy #5.         ; guarantee several FLAGs
EA02| A6 D2                 ldx xC          ; use X for low end of xC
EA04| D0**                 bne $0
EA06| C6 D3                 dec xC+1
EA04* 02
EA08| 88                 $0  dey             ; delay for the FLAGs
EA09| D0FD                 bne $0
EA0B|
EA0B| B1 D0                 lda @xP,y           ; get first byte
EA0D| 8D 0140               sta sccData         ;  and, start transmitting
EA10| A9 C0                 lda #0C0
EA12| 8D 0040               sta sccCtrl         ; reset TxUND
EA15| 80**                 bra $3          ; merge into loop to account for 1st byte
EA17|
EA17|                   ;** the main txFHDR loop here.
EA17| A9 04             $1  lda #TBE            ; faster loop
EA19| 2C 0040           $2  bit sccCtrl         ; wait for TBE
>>>>PAGE -  50  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


EA1C| F0FB                 beq $2
EA1E| B1 D0                 lda @xP,y           ; next byte
EA20| 8D 0140               sta sccData
EA23|
EA15* 0C
EA23| CA                 $3  dex             ; dec xC (done 1st to leave earlier)
EA24| D0**                 bne $4
EA26| C6 D3                 dec xC+1
EA28| 10**                 bpl $4
EA2A| 60                   rts             ; exit here at end of header
```

```
EA2B|
EA28* 01
EA24* 05
EA2B| C8                $4  iny             ; inc xP
EA2C| D0E9                  bne $1
EA2E| E6 D1                 inc xP+1
EA30| 80E5                  bra $1
EA32|
EA32|                   ;*************************************************************************
EA32|                   ;*  ckIdle, check for line idle for X * 100 µsec.  Note: we are assumed
EA32|                   ;* to be running w/ interrupts enabled; hence, we must check fType
EA32|                   ;* in case a frame comes in while polling.  C-flag is set if line is seen
EA32|                   ;* busy while we are waiting.
EA32|
EA32|                   ;*  ckIdle1 is an alternate entry to allow callers to "offset" delays
EA32|                   ;* which occur before ckIdle gets called.
EA32|                   ;*************************************************************************
E8E2* 32EA
E8D6* 32EA
E8C3* 32EA
EA32|                   ckIdle  ;** main entry point for full counts
EA32| A0 09                 ldy #9.         ; loop counter (~100 uSec worth)
E930* 34EA
EA34|                   ckIdle1 ;** alternate (user pre-loads Y)
EA34| AD 0040               lda sccCtrl         ; check for HUNT
EA37| 29 10                 and #HUNT
EA39| F0**                  beq $9          ; busy exit
EA3B| A5 AE                 lda fType
EA3D| D0**                  bne $9          ; something snuck in
EA3F| 88                    dey             ;  100 µsec?
EA40| D0F2                  bne ckIdle1         ; nope
EA42| CA                    dex             ;  yes, waited long enough?
EA43| D0ED                  bne ckIdle          ; nope
EA45| 18                    clc             ;  yes, set to show it's ok
EA46| 60                    rts
EA47|
EA3D* 08
EA39* 0C
EA47| 38                $9  sec             ; line is busy
EA48| 60                    rts
EA49|
EA49|                   ;*************************************************************************
EA49|                   ;*  bitCount, simple procedure to return # of 1's in A.
EA49|                   ;*************************************************************************
E86E* 49EA
E85A* 49EA
>>>>PAGE -  51  PCCARD    FILE: PCLAP.a65  PCcard - firmware for IBM-PC AppleTalk


EA49|                   bitCount    ;** bitCount, count # bits in A reg
EA49| 85 01                 sta t1      ; save for shifts
EA4B| A9 00                 lda #0      ; clear count
EA4D| 46 01             $1  lsr t1      ; next bit out
EA4F| F0**                  beq $2      ; done?
EA51| 69 00                 adc #0      ; nope, accumulate
EA53| D0F8                  bne $1
EA4F* 04
EA55| 69 00             $2  adc #0      ; add in last one
EA57| 60                    rts
EA58|
EA58|                   ;*************************************************************************
EA58|
EA58|                   ;*  Random, common routine to update xSeed w/ next psuedo-random value.
EA58|                   ;* note that t0/t1 and xSeed(+1) are treated as hi/lo bytes
EA58|                   ;*************************************************************************
E898* 58EA
EA58| A5 DF              Random  lda xSeed+1
EA5A| 85 00                 sta t0          ; t0 := xSeed * 256
EA5C| 85 01                 sta t1          ; (saves time below)
EA5E| 0A                    asl A
EA5F| 18                    clc
EA60| 65 00                 adc t0          ; A= xSeed * 768
EA62| 18                    clc
EA63| 65 DE                 adc xSeed
EA65| 85 00                 sta t0          ; t0,t1 := xSeed * 769
EA67|
EA67| 06 DF                 asl xSeed+1
EA69| 26 DE                 rol xSeed
EA6B| 06 DF                 asl xSeed+1
EA6D| 26 DE                 rol xSeed           ; xSeed := xSeed * 4
EA6F| 18                    clc
EA70| A5 01                 lda t1
EA72| 65 DF                 adc xSeed+1
EA74| 90**                  bcc $1
EA76| E6 DE                 inc xSeed
EA78| 18                    clc
EA74* 03
EA79| 69 07             $1  adc #7
```

```
EA7B| 85 DF                 sta xSeed+1
EA7D| A5 00                 lda t0
EA7F| 65 DE                 adc xSeed
EA81| 85 DE                 sta xSeed           ; xSeed := xSeed * 773 + 7
EA83| 60                    rts             ; leave upper byte ready to use
EA84|
EA84|
EA84|                   ;*******************************************************************************
EA84|                   ;**EOF  PCLAP
EA84|
EA84|                       .include    PCDDP       ; Directed Datagram Protocol handler
EA84|                   ;***    PCDDP, code to handle Directed Datagram Protocol stuff
EA84|
EA84|                   ;*******************************************************************************
EA84|                   ;*  xRXDDPX, receive DDP (eXtented) packet.
>>>>PAGE -  52  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


EA84|                   ;*******************************************************************************
EA84| A0 0F             xRXDDPX ldy #laData+xdDskt+2    ; index into header
EA86| B1 14                 lda @rtP,y          ; socket
EA88| 20 ****               jsr chkDDP          ; OK?
EA8B| B0**                 bcs $1
EA8D| 4C CDE4              jmp RCVtask         ; nope, error
EA90|
EA8B* 03
EA90| A0 11             $1  ldy #laData+xdType+2    ; if this is ATP
EA92| B1 14                 lda @rtP,y
EA94| C9 03                 cmp #ddATP
EA96| D0**                 bne $2
EA98| A9 10                 lda #fNATP          ; user want it?
EA9A| 24 30                 bit SysFlgs
EA9C| D0**                 bne $2          ; yes, handle as DDPX
EA9E| 4C ****              jmp xRXATPX         ;  else, process it
EAA1|
EA9C* 03
EA96* 09
EAA1| A0 10             $2  ldy #laData+xdData      ; front-end size
EAA3| 80**                 bra xRXDDP1
EAA5|
EAA5|                   ;*******************************************************************************
EAA5|                   ;*  xRXDDP, receive DDP (short) packet.  We perform special check here for
EAA5|                   ;* RTMP;  not needed in DDPX, as only short DDP is used for RTMP.
EAA5|                   ;*******************************************************************************
EAA5| A0 07             xRXDDP  ldy #laData+ddDskt+2    ; index into header
EAA7| B1 14                 lda @rtP,y          ; pickup socket
EAA9| C9 01                 cmp #rtmSKT         ; is this a special one?
EAAB|
EAAB| F0**                 beq $1
EAAD| 20 ****               jsr chkDDP          ; check socket entry
EAB0| B0**                 bcs $2          ; its ok, process rest
EAB2| 4C CDE4              jmp RCVtask         ;  and, ignore; just restart RCVtask
EAB5|
EAAB* 08
EAB5| 4C ****           $1  jmp xRXRTMP         ; its an RTMP socket, process it
EAB8|
EAB0* 06
EAB8| A0 09             $2  ldy #laData+ddType+2    ; if this is ATP
EABA| B1 14                 lda @rtP,y
EABC| C9 03                 cmp #ddATP
EABE| D0**                 bne $3
EAC0| A9 10                 lda #fNATP          ; user want it?
EAC2| 24 30                 bit SysFlgs
EAC4| D0**                 bne $3          ; yes, handle as DDP
EAC6| 4C ****               jmp xRXATP          ;  then, process it
EAC9|
EAC4* 03
EABE* 09
EAC9| A0 08             $3  ldy #laData+ddData      ; length of DMAinfo packet
EACB|
EAA3* 26
EACB|                   xRXDDP1                 ;** entry for xRXDDP/xRXDDPX
EACB| A2 00                 ldx #0          ; RID, 0 for DDP
>>>>PAGE -  53  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


EACD| A9 24                 lda #024            ; else, Host FC for rxDDP
EACF| 20 ****               jsr RXDDP1          ; use common code
EAD2| 4C CAE4              jmp RCVfree         ;  and, merge
EAD5|
EAD5|                   ;*******************************************************************************
EAD5|                   ;*  RXDDP1, acquire DMA and transfer our packet data.
EAD5|                   ;* A has Host FC,  Y has front-end length,  X has RID (0 for RXDDP).
EAD5|                   ;* Note: we return to caller for post-transfer stuff (ATP).
EAD5|                   ;*******************************************************************************
EAD0* D5EA
EAD5|                   RXDDP1                  ;** entry from DDP/ATP
EAD5| DA                   phx             ; save params
```

```
EAD6| 48                    pha
EAD7| 5A                    phy
EAD8| 84 00                 sty t0          ; save for computation
EADA| A0 01
EADC| 38                    sec
EADD| B2 14                 lda @rtP
EADF| E5 00                 sbc t0          ; (minus front-end)
EAE1| 48                    pha             ; save for dC
EAE2| B1 14                 lda @rtP,y
EAE4| E9 00                 sbc #0
EAE6| 48                    pha
EAE7|
EAE7| A2 74                 ldx #DMAlock        ; acquire DMA
EAE9| 20 CCE1               jsr semaP
EAEC| A9 C0                 lda #DMAinfo        ; tell what we want
EAEE| 20 24E4               jsr waitDMA
EAF1|
EAF1|                     ;** now, setup for DMA transfer, and do front-end
EAF1| 68                    pla             ; setup dC for later
EAF2| 85 1B                 sta dC+1
EAF4| 68                    pla
EAF5| 85 1A                 sta dC
EAF7| FA                    plx             ; front-end looper
EAF8| 68                    pla             ; FC
EAF9| 8D 0080               sta DMAreg
EAFC| 68                    pla             ; RID (0 for DDP)
EAFD| 8D 0080               sta DMAreg
EB00|
EB00|                     ;** copy header during DMAinfo state..
EB00| A0 02                 ldy #2          ; skip over rtP
EB02| B1 14               $1  lda @rtP,y
EB04| 8D 0080               sta DMAreg
EB07| C8                    iny
EB08| CA                    dex
EB09| D0F7                 bne $1
EB0B|
EB0B|                     ;** now, setup dP for common path
EB0B| A5 1A                 lda dC          ; check if any DMAdata state needed
EB0D| 05 1B                 ora dC+1
EB0F| F0**                  beq $9          ;  if not, skip it
EB11|
EB11| 18                    clc
EB12| 98                    tya             ; else, set dP
>>>>PAGE -  54  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


EB13| 65 14                 adc rtP
EB15| 85 18                 sta dP
EB17| A5 15                 lda rtP+1
EB19| 85 19                 sta dP+1
EB1B|
EB1B|                     ;** now, copy the packet (note: dC has already been setup)
EB1B| A9 D0                 lda #DMAdata        ; show ready for data now
EB1D| 20 24E4               jsr waitDMA
EB20|
EB20| 20 8DE4               jsr wrDMA           ; do it
EB23|
EB0F* 12
EB23| 60                  $9  rts             ;  and, we're ready for more
EB24|
EB24|                     ;*************************************************************************
EB24|                     ;*  xOPNSKT, open (register) a socket #.  (Host FC=$21).
EB24|                     ;*************************************************************************
EB24|
EB24| AD 0080             xOPNSKT lda DMAreg          ; pickup user param
EB27| 20 ****                 jsr chkSKT          ; lookup
EB2A| B0**                   bcs OPNCLSe         ; error
EB2C| 1D 4001                ora SKTtbl,x        ; else, add it
EB2F|
EB2F| 9D 4001             OPNCLSx sta SKTtbl,x        ; update table byte
EB32| A9 00                  lda #0          ; show no error
EB34| 4C 45E3                jmp CMDdone         ;  and, merge
EB37|
EB37|                     ;*************************************************************************
EB37|                     ;*  xCLSSKT, close (un-register) a socket #.  (Host FC=$22)
EB37|                     ;*************************************************************************
EB37| AD 0080             xCLSSKT lda DMAreg          ; get param
EB3A| 20 ****                jsr chkSKT          ; is it open?
EB3D| 90**                   bcc OPNCLSe         ; no, error
EB3F| 5D 4001                eor SKTtbl,x        ; yes, remove it
EB42| 80EB                   bra OPNCLSx
EB44|
EB3D* 05
EB2A* 18
EB44| A9 01             OPNCLSe lda #SKTerr         ; error processing
EB46| 4C 45E3                jmp CMDdone
EB49|
EB49|                     ;*************************************************************************
```

```
EB49|                   ;*  chkDDP, check DDP length, then fall thru to chkSKT.  C is clear if
EB49|                   ;* length is not OK (or if chkSKT doesn't find socket).
EB49|                   ;*****************************************************************************
EAAE* 49EB
EA89* 49EB
EB49| 48                chkDDP  pha             ; save socket# for chkSKT
EB4A| A0 00                 ldy #0          ; compute LAP-3
EB4C| 38                    sec
EB4D| B1 14                 lda @rtP,y
EB4F| C8                    iny
EB50| E9 03                 sbc #3
EB52| 85 00                 sta t0
EB54| B1 14                 lda @rtP,y
>>>>PAGE -  55  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


EB56| E9 00                 sbc #0
EB58| 85 01
EB5A|
EB5A| A0 05                 ldy #laData+ddLng+2     ; check on DDP length vs LAP length
EB5C| B1 14                 lda @rtP,y
EB5E| C8                    iny
EB5F| 29 03                 and #003            ; mask hop
EB61| C5 01                 cmp t0+1            ; check it'
EB63| D0**                  bne $69
EB65| B1 14                 lda @rtP,y
EB67| C5 00                 cmp t0
EB69| F0**                  beq $8          ; continue if OK
EB6B|
EB63* 06
EB6B| 68                $69 pla             ; pop saved param
EB6C| 18                    clc             ; show, not-OK
EB6D| 60                    rts             ;  and, return to caller
EB6E|
EB69* 03
EB6E| A9 08              $8  lda #fADDP          ; accept all DDP?
EB70| 24 30                 bit SysFlgs
EB72| F0**                  beq $9          ; no, proceed to check SKT
EB74| 68                    pla
EB75| 38                    sec             ; else, show OK
EB76| 60                    rts
EB77|
EB72* 03
EB77| 68                $9  pla             ; pull socket# for chkSKT if ok
EB78|
EB78|                   ;*****************************************************************************
EB78|                   ;*  chkSKT, lookup socket (in A), does BIT here, leaves Z set.
EB78|                   ;*****************************************************************************
EB3B* 78EB
EB28* 78EB
EB78| A8                chkSKT  tay             ; save socket
EB79| 4A                    lsr A           ; shift off offset
EB7A| 4A                    lsr A
EB7B| 4A                    lsr A
EB7C| AA                    tax             ; X is offset into SKTtbl
EB7D| 98                    tya             ; get protocol back
EB7E| 29 07                 and #007            ; bit#
EB80| A8                    tay
EB81| B9 4BE0               lda BITS,y          ; get bit mask
EB84|
EB84| 18                    clc             ; assume not there
EB85| 3C 4001               bit SKTtbl,x        ; test current bit
EB88| F0**                  beq $9          ; exit if not
EB8A|
EB8A| 38              $8  sec             ;  else, show present
EB88* 01
EB8B| 60                $9  rts
EB8C|
EB8C|                   ;*****************************************************************************
EB8C|
EB8C|                   ;*  xTXDDP, transmit DDP packet.  (Host FC=$23)  The single entry reads
>>>>PAGE -  56  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


EB8C|                   ;* a packet formatted to assume DDPX header.  If checks prove that the node
EB8C|                   ;* exists on our link (the normal case!), the header will be modified to
EB8C|                   ;* that of a DDP.
EB8C|                   ;*  Note, a code of $27 acts as $23, except that the transmission is
EB8C|                   ;* retried.  This is designed to help support NBP.
EB8C|                   ;*  Also, a code of $2B acts as $23, except that the checksum field will
EB8C|                   ;* be filled in.
EB8C|                   ;*****************************************************************************
EB8C|                   xSNDNBP     ;*** alternate name for NBP command
EB8C| A5 CC                 lda nbTTP           ; one in progress?
EB8E| F0**                  beq xTXDDP          ; no, continue
EB90|
EB90| A9 FF              SNDNBPe lda #REQerr         ; flag error
EB92| 4C 45E3               jmp CMDdone
```

```
EB95|
EB8E* 05
EB95|                    xTXDDP                  ;** entry from CMDtask
EB95| 20 85E5                jsr xTXPKTA         ; allocate buffer
EB98| A5 18                  lda dP          ; save copy of buffer
EB9A| 85 04                  sta t4
EB9C| A5 19                  lda dP+1
EB9E| 85 05                  sta t4+1
EBA0| A0 05                  ldy #laData+xdCKS       ; copy stuff for our assumed long header
EBA2| A9 00                  lda #0          ; assume no checksum, zero xdCKS
EBA4| 91 02                  sta @t2,y
EBA6| C8                    iny
EBA7| 91 02                  sta @t2,y
EBA9| C8                    iny
EBAA| AD 0080                lda DMAreg          ; xdDnet
EBAD| 91 02                  sta @t2,y
EBAF| C8                    iny
EBB0| AD 0080                lda DMAreg
EBB3| 91 02                  sta @t2,y
EBB5| C8                    iny
EBB6| A5 0C                  lda ThisNet         ; xdSnet
EBB8|
EBB8| 91 02                  sta @t2,y
EBBA| C8                    iny
EBBB| A5 0D                  lda ThisNet+1
EBBD| 91 02                  sta @t2,y
EBBF| C8                    iny
EBC0| AD 0080                lda DMAreg          ; xdDnode
EBC3| 91 02                  sta @t2,y
EBC5| C8                    iny
EBC6| A5 69                  lda myNode          ; xdSnode (me!!)
EBC8| 91 02                  sta @t2,y
EBCA| C8                    iny
EBCB| AD 0080                lda DMAreg          ; xdDskt
EBCE| 91 02                  sta @t2,y
EBD0| C8                    iny
EBD1| AD 0080                lda DMAreg          ; xdSskt
EBD4| 91 02                  sta @t2,y
EBD6| C8                    iny
EBD7| AD 0080                lda DMAreg          ; xdType
EBDA| 91 02                  sta @t2,y
>>>>PAGE -  57  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


EBDC|
EBDC| AD 0080                lda DMAreg          ; DDP data size
EBDF| 85 1A                  sta dC          ; save for rdDMA
EBE1| AD 0080                lda DMAreg
EBE4| 85 1B                  sta dC+1
EBE6|
EBE6|                    ;** if its a $27, pickup interval and tries
EBE6| A5 12                  lda pcREQ           ; normal?
EBE8| C9 27                  cmp #027            ; TXNBP?
EBEA| D0**                  bne $0          ;  no, skip
EBEC|
EBEC| AD 0080                lda DMAreg          ; else, pickup extra data
EBEF| 85 CD                  sta nbTmOut         ; interval
EBF1| F09D                  beq SNDNBPe         ; error if invalid
EBF3| AD 0080                lda DMAreg
EBF6| C9 FF                  cmp #0FF            ; check on "infinite"
EBF8| F096                  beq SNDNBPe
EBFA| 85 CE                  sta nbRtrys         ; tries
EBFC|
EBFC|                    ;** check on correct type (use DDP/DDPX)
EBEA* 10
EBFC| A0 07              $0  ldy #laData+xdDnet
EBFE| B1 02                  lda @t2,y
EC00| C8                    iny
EC01| 11 02                  ora @t2,y
EC03| F0**                  beq TXDDP           ; if Dnet is 0, use short
EC05|
EC05|                    ;** check on validity of aBridge & ThisNet
EC05| A5 0B                  lda aBridge         ; bridge present?
EC07| D0**                  bne $1          ;  if so, ThisNet must be valid
EC09| A5 0C                  lda ThisNet
EC0B| 05 0D                  ora ThisNet+1
EC0D| F0**                  beq TXDDPX          ; if null net, send long one
EC0F|
EC07* 06
EC0F| B1 02              $1  lda @t2,y
EC11| C5 0D                  cmp ThisNet+1       ; else, it is same as me?
EC13| D0**                  bne TXDDPX          ;  whoops, use long (fill in rest)
EC15|
EC15| 88                    dey
EC16| B1 02                  lda @t2,y
EC18| C5 0C                  cmp ThisNet
EC1A| F0**                  beq TXDDP
EC1C|
```

```
EC1C|                   ;** finish building TXDDPX header
EC13* 07
EC0D* 0D
EC1C| A5 0B             TXDDPX  lda aBridge
EC1E| D0**                  bne $1          ; if valid, use it
EC20| A0 0B                 ldy #laData+xdDnode     ;  else, from caller's
EC22| B1 02                 lda @t2,y
EC24|
EC1E* 04
EC24| A0 00             $1  ldy #laDst
EC26| 91 02                 sta @t2,y
>>>>PAGE -  58  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


EC28| A9 0D                 lda #xdData         ; size of DDP header
EC2A| A2 02                 ldx #laDDPX         ; type of header
EC2C| 80**                  bra TXDDP1          ;  and, share code
EC2E|
EC2E|                   ;** whoopee, we can use short data, so fill in the rest of the stuff
EC1A* 12
EC03* 29
EC2E| A0 0F             TXDDP   ldy #laData+xdType      ; copy common stuff to stack
EC30| B1 02                 lda @t2,y           ; xdType
EC32| 88                   dey
EC33| 48                   pha
EC34| B1 02                 lda @t2,y           ; xdSskt
EC36| 88                   dey
EC37| 48                   pha
EC38| B1 02                 lda @t2,y           ; xdDskt
EC3A| 88                   dey
EC3B| 48                   pha
EC3C| 88                   dey
EC3D| B1 02                 lda @t2,y           ; xdDnode (our locat laDst)
EC3F|
EC3F|                   ;** build LAP/DDP header (partially)
EC3F| A0 00                 ldy #laDst
EC41| 91 02                 sta @t2,y
EC43| A0 05                 ldy #laData+ddDskt      ; copy sockets, type
EC45| 68                   pla
EC46| 91 02                 sta @t2,y           ; ddDskt
EC48| C8                   iny
EC49| 68                   pla
EC4A| 91 02                 sta @t2,y           ; ddSskt
EC4C| C8                   iny
EC4D| 68                   pla
EC4E| 91 02                 sta @t2,y           ; ddType
EC50| A9 05                 lda #ddData         ; header size
EC52| A2 01                 ldx #laDDP          ;    "   type
EC54|
EC54|                   ;** adjust dC for our remaining (DDP data packet) length
EC2C* 26
EC54|                   TXDDP1                  ;** common path for TXDDP/TXDDPX
EC54| 85 00                 sta t0          ; save header size
EC56| A0 01                 ldy #laSrc
EC58| A5 69                 lda myNode          ; complete header
EC5A| 91 02                 sta @t2,y           ; laSrc (myNode)
EC5C|
EC5C| C8                   iny
EC5D| 8A                   txa
EC5E| 91 02                 sta @t2,y           ; laType
EC60| C8                   iny
EC61| 18                   clc
EC62| A5 1A                 lda dC
EC64| 65 00                 adc t0          ; adjust by DDP header size
EC66| AA                   tax
EC67| A5 1B                 lda dC+1
EC69| 69 00                 adc #0
EC6B| 91 02                 sta @t2,y           ; ddLng
EC6D| C8                   iny
>>>>PAGE -  59  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


EC6E| 8A                   txa
EC6F| 91 02                 sta @t2,y
EC71|
EC71|                   ;** setup params to xTXPKTX and merge
EC71| 18                   clc
EC72| A5 00                 lda t0          ; A= DDP header size
EC74| 69 03                 adc #laData         ; A= LAP + DDP header size
EC76| 48                   pha             ; save total length
EC77|
EC77|                   ;** check if this is $23 (TXDDP) or $27 (TXNBP)
EC77| A5 12                 lda pcREQ
EC79| C9 27                 cmp #027
EC7B| D0**                  bne $1
EC7D| 4C ****               jmp TXNBP           ; do it w/ retries
EC80|
EC80|                   ;*****************************************************************************
```

```
EC80|                   ;*  code copied from xTXPKT.  We do it here so that we can decide to
EC80|                   ;* compute checksum after acquiring the DDP packet's data
EC80|                   ;*****************************************************************************
EC7B* 03
EC80| 68                $1  pla             ; else, normal (1-shot)
EC81| A2 40                 ldx #CMDtcb         ; set param for xTXPKTD
EC83| 20 2BE8               jsr xTXPKTD         ; acquire the packet's data
EC86|
EC86|                   ;** check on whether CKS is desired.
EC86| A5 12                 lda pcREQ           ; pickup function code
EC88| C9 25                 cmp #025            ; set?
EC8A| D0**                  bne $2          ;  nope, skip it
EC8C| A0 02                 ldy #laType
EC8E| B1 02                 lda @t2,y           ; is it DDPX?
EC90| C9 02                 cmp #laDDPX
EC92| F0**                  beq doCKS           ; yes, compute it
EC94|
EC8A* 08
EC94| 4C ****           $2  jmp TXDDP2          ; else, skip it
EC97|
EC97|
EC97|                   ;*****************************************************************************
EC97|                   ;*  Compute the CKS over our header, etc.  Note that because we can take
EC97|                   ;* a while to do the computation, we will let other tasks have a crack at
EC97|                   ;* the system.
EC97|                   ;*  During the calculation:
EC97|                   ;*      t0 is CKS
EC97|                   ;*      t2 is header ptr
EC97|                   ;*      t4 is data ptr
EC97|                   ;*      t6 is data lng
EC97|                   ;*****************************************************************************
EC92* 03
EC97| 20 F7E1         doCKS   jsr Yield           ; and, give up control
EC9A| A5 44                 lda CMDBufN         ; compute addresses agin
EC9C| A2 04                 ldx #t4
EC9E| 20 44E2               jsr cvtBufN
ECA1| 18                    clc             ; set t2 = header
ECA2| A5 04                 lda t4
ECA4| 69 68                 adc #HDROFS % 0100
>>>>PAGE -  60  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


ECA6| 85 02                 sta t2
ECA8| A5 05                 lda t4+1
ECAA| 69 02                 adc #HDROFS / 0100
ECAC| 85 03                 sta t2+1
ECAE|
ECAE|                   ;** start checksum over DDP header
ECAE| 64 00                 stz t0          ; CKS.hi = 0
ECB0| A2 09                 ldx #xdData-xdDnet      ; size of header
ECB2| A9 00                 lda #0          ; CKS.lo = 0
ECB4| A0 07                 ldy #laData+xdDnet      ; offset to start checksum at
ECB6|
ECB6|                   ckHdr   ;** header checksum loop, A=CKS.lo
ECB6| 18                $1  clc
ECB7| 71 02                 adc @t2,y           ; next header byte
ECB9| 90**                  bcc $2
ECBB| E6 00                 inc t0          ; carry out to CKS.hi
ECB9* 02
ECBD| 0A                $2  asl A
ECBE| 26 00                 rol t0
ECC0| 90**                  bcc $3
ECC2| 1A                    ina
ECC0* 01
ECC3| C8                $3  iny             ; adjust ptr
ECC4| CA                    dex             ; count down
ECC5| D0EF                  bne $1
ECC7| 85 01                 sta t1          ; save CKS.lo
ECC9|
ECC9|                   ;** setup ptrs, etc. for data stuff
ECC9|
ECC9| A0 02                 ldy #2
ECCB| B1 04                 lda @t4,y           ; get data count
ECCD| C8                    iny
ECCE| 85 06                 sta t6
ECD0| B1 04                 lda @t4,y
ECD2| 85 07                 sta t6+1
ECD4|
ECD4|                   ;** setup rest of ckData loop
ECD4| 18                $4  clc
ECD5| A5 04                 lda t4
ECD7| 69 04                 adc #4          ; ptr to data
ECD9| 85 04                 sta t4
ECDB| A6 06                 ldx t6          ; count (low)
ECDD| D0**                  bne ckData          ; enter odd loop
ECDF| C6 07                 dec t6+1            ;  else, adjust for even 256
ECE1| 10**                  bpl ckData0         ; if not null, enter 256 count loop
ECE3| A5 01                 lda t1          ;  else, reload CKS.lo
```

```
ECE5| 80**                  bra stCKS           ;   and, skip to store (null data)
ECE7|
ECDD* 08
ECE7|                   ckData  ;** data checksum loop for mod 256 odd counts
ECE7| A0 00                 ldy #0          ; make sure of index
ECE9| A5 01                 lda t1          ; re-load low-byte of current CKS
ECEB| 18                $1  clc             ; the loop!!!
ECEC| 71 04                 adc @t4,y           ; add value
ECEE| 90**                  bcc $2
>>>>PAGE -  61  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


ECF0| E6 00                 inc t0
ECEE* 02
ECF2| 0A                $2  asl A           ; rotate it
ECF3| 26 00                 rol t0
ECF5| 90**                  bcc $3
ECF7| 1A                    ina
ECF5* 01
ECF8| C8                $3  iny
ECF9| CA                    dex
ECFA| D0EF                  bne $1
ECFC| C6 07                 dec t6+1            ; any more?
ECFE| 30**                  bmi stCKS           ;  no, simply store here
ED00| 85 01                 sta t1          ; save CKS.lo
ED02|
ED02|                   ;** having taken care of odd counts, adjust t4 and entry 256 count loop
ED02| 18                    clc             ; adjust t4 for 256 count loop
ED03| A5 04                 lda t4          ; current ptr
ED05| 65 06                 adc t6          ; add offset handled above
ED07| 85 04                 sta t4
ED09| 90**                  bcc ckData0         ; skip if we didn't overflow
ED0B| E6 05                 inc t4+1            ;  else, adjust hi byte
ED0D|
ED09* 02
ECE1* 2A
ED0D|                   ckData0 ;** data checksum loop when we have taken care of odd front end.
ED0D|
ED0D| A0 00                 ldy #0          ; make sure of index
ED0F| A5 01                 lda t1          ; re-load CKS.lo
ED11| 18                $1  clc             ; the loop!!!
ED12| 71 04                 adc @t4,y           ; add value
ED14| 90**                  bcc $2
ED16| E6 00                 inc t0
ED14* 02
ED18| 0A                $2  asl A           ; rotate it
ED19| 26 00                 rol t0
ED1B| 90**                  bcc $3
ED1D| 1A                    ina
ED1B* 01
ED1E| C8                $3  iny             ; advance ptr (also loop counter!)
ED1F| D0F0                  bne $1
ED21| E6 05                 inc t4+1            ; adjust carry over
ED23| C6 07                 dec t6+1            ; major loop counter
ED25| 10EA                  bpl $1
ED27|
ED27|                   ;** checksum is done;  A=CKS.lo, t0=CKS.hi store it
ECFE* 27
ECE5* 40
ED27| A0 06              stCKS   ldy #laData+xdCKS+1
ED29| 91 02                 sta @t2,y           ; CKS.lo
ED2B| 88                    dey
ED2C| A5 00                 lda t0
ED2E| 91 02                 sta @t2,y           ; CKS.hi
ED30|
ED30|                   ;*************************************************************************
EC95* 30ED
>>>>PAGE -  62  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


ED30|                   TXDDP2  ;* queue the packet and wait for it to go out..
ED30| A5 44                 lda CMDBufN
ED32| 64 40                 stz CMDFlgs         ; clear other flags
ED34| 20 80E2               jsr putXmit
ED37|
ED37| A9 04                 lda #sTXDONE        ; show what we're waiting on
ED39| 20 64E1               jsr Wait
ED3C|
ED3C| A5 44                 lda CMDBufN         ; get our buffer number back
ED3E| 20 73E2               jsr putFree         ; add it back to free list
ED41| 4C 4CE3               jmp CMDstat         ;  and, we're done...
ED44|
ED44|                   ;*************************************************************************
ED44|                   ;*  TXNBP - This section handles the setting up of ATP block.  The rest
ED44|                   ;* of the processing for SNDNBP is performed by ATP task.
ED44|                   ;* NOTE: THE FOLLOWING CODE WAS ADDED DURING -B ECO.
ED44|
ED44|                   ;*************************************************************************
```

```
EC7E* 44ED
ED44|                   TXNBP               ;*** entry upon recognizing NBP type send
ED44| A6 E8                 ldx lstTTP      ; start from last one allocated.
ED46| BD 0002           $1  lda ttState,x   ; this one busy?
ED49| F0**                 beq txn1        ;  no, we have one
ED4B| 8A                   txa         ; else, advance thru table
ED4C| 18                   clc
ED4D| 69 0D                adc #TTSIZE     ;  by incrementing by table entry size
ED4F| D0**                 bne $2      ; skip if not at page boundary
ED51| A9 09                lda #TTBASE % 0100  ; else, start at front
ED4F* 02
ED53| AA                $2  tax
ED54| C5 E8                 cmp lstTTP      ; have we gone around all the way?
ED56| D0EE                 bne $1      ;  no, keep looking
ED58|
ED58|                   ;** we haven't found a free one; signal error to CMD caller.
ED58| 68                   pla         ; pop header size
ED59| A9 FE                lda #NoFreeErr
ED5B| 4C 45E3              jmp CMDdone     ;  and, thats it
ED5E|
ED5E|                   ;** having found a table, setup the request stuff.
ED49* 13
ED5E| 86 E8             txn1    stx lstTTP          ; save for others
ED60| 86 CC                 stx nbTTP           ;  and us, later
ED62| A9 46                 lda #SENDNBPn       ; its state (temp, to hold)
ED64| 9D 0002               sta ttState,x
ED67| A5 47                 lda CMDRID          ; our request ID
ED69| 9D 0102               sta ttRID,x
ED6C| A5 CD                 lda nbTmOut
ED6E| 9D 0502               sta ttTmOut,x
ED71| A5 CE                 lda nbRtrys
ED73| 9D 0602               sta ttRtrys,x
ED76| A5 44                 lda CMDBufN         ; the buffer to use
ED78| 9D 0402               sta ttBufN,x
ED7B|
ED7B|                   ;** now, acquire rest of this data
ED7B| A2 58                 ldx #ATPtcb         ; param for buffer
>>>>PAGE -  63  PCCARD    FILE: PCDDP.a65  PCcard - firmware for IBM-PC AppleTalk


ED7D|
ED7D| 68                   pla             ; length of header
ED7E| 20 2BE8               jsr xTXPKTD         ; use shared code
ED81|
ED81| A6 CC                 ldx nbTTP           ; get our ptr back
ED83| A9 40                 lda #SENDNBP        ; now we get correct start
ED85| 9D 0002               sta ttState,x
ED88| 86 E9                 stx lstATP          ; start up ATP
ED8A| 64 44                 stz CMDBufN
ED8C| 64 47                 stz CMDRID
ED8E| 4C 45E3               jmp CMDdone         ;  and, that's all we do here
ED91|
ED91|                   ;*****************************************************************************
ED91|                   ;** end of PCDDP
ED91|
ED91|                       .include    PCRTMP      ; Routing Table Maintenance Protocol
ED91|                   ;**  PCRTMP, handler for Routing Table Maintenance
ED91|                   ;*****************************************************************************
ED91|                   ;*  This little stub handles reception of RTMP packets.  This version is
ED91|                   ;* only applicable for non-Bridge nodes.  All we do is pickup values for
ED91|                   ;* aBridge and ThisNet.
ED91|                   ;*  Note:  all of this module is new with the -A rev of ROM.
ED91|                   ;*****************************************************************************
EAB6* 91ED
ED91|                   xRXRTMP                 ;** entry from xRXDDP
ED91| A0 09                ldy #laData+ddType+2    ; look at its type
ED93| B1 14                 lda @rtP,y
ED95| C9 01                 cmp #ddRTMP         ; is this RTMP response?
ED97| D0**                 bne $69         ;  if not, flush it
ED99|
ED99|                   ;** this is an RTMP response, so copy the network # and bridge #
ED99| A0 03                ldy #laSrc+2        ; source is bridge's id
ED9B| B1 14                 lda @rtP,y
ED9D| 85 0B                sta aBridge
ED9F| A0 0A                ldy #laData+ddData+rtmNet+2 ; copy net# from packet
EDA1| B1 14                 lda @rtP,y
EDA3| C8                   iny
EDA4| 85 0C                sta ThisNet
EDA6| B1 14                 lda @rtP,y
EDA8| 85 0D                sta ThisNet+1
EDAA|
EDAA|                   ;** reload the RTMP timer
EDAA| A9 5A                lda #90.            ; loooong timer
EDAC| 85 1E                sta RTMPtmr
EDAE|
ED97* 15
EDAE| 4C CDE4           $69 jmp RCVtask         ; then, toss the packet
EDB1|
```

```
EDB1|                   ;*****************************************************************************
EDB1|                   ;*  setRTMP, a routine called by setLAP to send the RTMP+ packets.  We
EDB1|
EDB1|                   ;* locate code here to modularize the RTMP stuff.
EDB1|                   ;*****************************************************************************
EDB1|                   setRTMP                 ;** jsr'd by setLAP
EDB1| A2 18                 ldx #dP          ; allocate a buffer (if we can)
>>>>PAGE -  64  PCCARD    FILE: PCRTMP.a65  PCcard - firmware for IBM-PC AppleTalk


EDB3| 20 33E2               jsr getFree
EDB6| D0**                 bne $1           ; if it went OK, process it
EDB8| A5 69                 lda myNode           ; else, simply return my node #
EDBA| 4C 45E3               jmp CMDdone
EDBD|
EDBD|                   ;** make up the RTMP+ packet
EDB6* 05
EDBD| 85 44              $1  sta CMDBufN          ; save the buffer #
EDBF| 18                   clc
EDC0| A5 18                 lda dP
EDC2| 69 68                 adc #HDROFS % 0100
EDC4| 85 02                 sta t2
EDC6| A5 19                 lda dP+1
EDC8| 69 02                 adc #HDROFS / 0100
EDCA| 85 03                 sta t2+1
EDCC|
EDCC|                   ;** build an RTMP+ packet (only using header area)
EDCC| A0 00                 ldy #0
EDCE| A9 FF                 lda #0FF             ; broadcast
EDD0| 91 02                 sta @t2,y
EDD2| C8                   iny
EDD3| A5 69                 lda myNode
EDD5| 91 02                 sta @t2,y
EDD7| C8                   iny
EDD8| A9 01                 lda #laDDP
EDDA| 91 02                 sta @t2,y
EDDC| C8                   iny
EDDD| A9 00                 lda #0           ; length
EDDF| 91 02                 sta @t2,y
EDE1| C8                   iny
EDE2| A9 06                 lda #6
EDE4| 91 02                 sta @t2,y
EDE6| C8                   iny
EDE7| A9 01                 lda #rtmSKT
EDE9| 91 02                 sta @t2,y
EDEB| C8                   iny
EDEC| 91 02                 sta @t2,y
EDEE| C8                   iny
EDEF| A9 05                 lda #ddRTMP1
EDF1| 91 02                 sta @t2,y
EDF3| C8                   iny
EDF4| A9 01                 lda #rtmCMD1
EDF6| 91 02                 sta @t2,y
EDF8| C8                   iny
EDF9|
EDF9|                   ;** prepare transmission
EDF9| 98                   tya              ; our length
EDFA| A0 00                 ldy #0
EDFC| 91 18                 sta @dP,y            ; header's length
EDFE| C8                   iny
EDFF| A9 40                 lda #CMDtcb          ; who to wake-up
EE01| 91 18                 sta @dP,y            ; data length (nil)
EE03| C8                   iny
EE04| A9 00                 lda #0
EE06| 91 18                 sta @dP,y
>>>>PAGE -  65  PCCARD    FILE: PCRTMP.a65  PCcard - firmware for IBM-PC AppleTalk


EE08| C8                   iny
EE09| 91 18                 sta @dP,y
EE0B|
EE0B|                   ;** the transmit loop
EE0B| A0 03                 ldy #3           ; times to try
EE0D|
EE0D| 5A                rtXmit  phy              ; save counter
EE0E| 64 40                 stz CMDflgs
EE10| A5 44                 lda CMDBufN
EE12| 20 80E2               jsr putXmit          ; start it out
EE15| A9 04                 lda #sTXDONE         ;  and, wait for it
EE17| 20 64E1               jsr Wait
EE1A|
EE1A| A0 01                 ldy #1           ; 1 second delay
EE1C| 5A                $1  phy              ; save secs ctr
EE1D| A5 1C                 lda TMRsecs
EE1F| 48                $2  pha              ; wait for a second
EE20| 20 F7E1               jsr Yield
EE23| 68                   pla
EE24| C5 1C                 cmp TMRsecs
```

```
EE26| F0F7                beq $2
EE28| 7A                  ply             ; check if enuff gone bye...
EE29| 88                  dey
EE2A| D0F0                bne $1
EE2C|
EE2C|                   ;** having waited, check if any reply came in
EE2C| 7A                  ply             ; get counter in case of success
EE2D| A5 0B               lda aBridge         ; has one responded?
EE2F| D0**                bne $8          ;  yes, we're done
EE31| 88                  dey             ; else, do a few times
EE32| D0D9                bne rtXmit
EE34|
EE2F* 03
EE34| A5 44              $8  lda CMDBufN         ; done, give buffer back
EE36| 20 73E2              jsr putFree
EE39|
EE39|                   ;** following was moved from xINITLAP
EE39| A5 69              $9  lda myNode          ; our number is status!!
EE3B| 4C 45E3              jmp CMDdone         ;  and, back to IdleLp
EE3E|
EE3E|                   ;*****************************************************************************
EE3E|                   ;***    EOF, of PCRTMP
EE3E|
EE3E|
EE3E|                       .include    PCNBP       ; Name Binding Protocol handler
EE3E|                   ;***    PCNBP, support for Name Binding on PCcard
EE3E|                   ;*****************************************************************************
EE3E|                   ;*  As part of -A changes, we have removed the NBPlkup.  Instead, it is
EE3E|                   ;* replaced by a retry facility in DDP.  Thus, the code formerly contained
EE3E|                   ;* in this module is now at end of PCDDP.
EE3E|                   ;*****************************************************************************
EE3E|
EE3E|                   ;*****************************************************************************
EE3E|                   ;***    end of PCNBP
EE3E|
>>>>PAGE -  66  PCCARD    FILE:   PCcard - firmware for IBM-PC AppleTalk


EE3E|
EE3E|                       .include    PCATP       ; Apple Transaction Protocol handler
EE3E|                   ;****   PCATP, handler for AppleTalk Transaction Protocol
EE3E|                   ;*****************************************************************************
EE3E|                   ;*  AppleTalk Transaction Protocol handler.  Unlike the handlers for the
EE3E|                   ;* other protocols, most of this runs as an independent task.  Transmission
EE3E|                   ;* of Request and Response packets is initiated by this task, rather than
EE3E|                   ;* by the CMDtask.  The one-second time maintained by PCEXEC is used for all
EE3E|                   ;* timeouts required by the protocol.
EE3E|                   ;*  Each transaction is kept track of by a table entry (the Transaction
EE3E|                   ;* Table).  These table entries are created by CMDtask, but maintained by
EE3E|                   ;* ATPtask.
EE3E|                   ;*****************************************************************************
EE3E|
EE3E|                   ;*****************************************************************************
EE3E|                   ;*  xRXATPX, received an (DDP extended) ATP packet;  do something with it...
EE3E|                   ;*****************************************************************************
EE3E|                   xRXATPX                 ;** entry from RCVtask
EE3E| A2 7C                ldx #ATPlock        ; lock now to make sure we play w/ vars
EE40|
EE40| 20 CCE1              jsr semaP
EE43| A0 14                ldy #laData+xdData+atTID+2  ; setup tempts for fndTT;  src TID
EE45| B1 14                lda @rtP,y
EE47| 85 00                sta t0
EE49| C8                  iny
EE4A| B1 14                lda @rtP,y
EE4C| 85 01                sta t1
EE4E| A0 0E                ldy #laData+xdSNode+2
EE50| B1 14                lda @rtP,y
EE52| 85 02                sta t2          ; src node
EE54| A0 10                ldy #laData+xdSskt+2
EE56| B1 14                lda @rtP,y
EE58| 85 03                sta t3          ; src socket
EE5A| A0 0B                ldy #laData+xdSnet+2
EE5C| B1 14                lda @rtP,y
EE5E| 85 04                sta t4          ; src net
EE60| C8                  iny
EE61| B1 14                lda @rtP,y
EE63| 85 05                sta t5
EE65|
EE65| A0 13                ldy #laData+xdData+atBtMap+2    ; setup common code
EE67| B1 14                lda @rtP,y          ; bitMap (seq nbr)
EE69| 85 06                sta t6
EE6B| 88                  dey
EE6C| A9 18                lda #laData+xdData+atData   ; length of header
EE6E| 80**                bra RXATP           ;  and, merge
EE70|
EE70|                   ;*****************************************************************************
EE70|                   ;*  xRXATP, received an ATP packet;  do something with it...
EE70|                   ;*****************************************************************************
```

```
EE70|                   xRXATP              ;** entry from RCVtask
EE70| A2 7C                 ldx #ATPlock        ; lock now to make sure we play w/ vars
EE72| 20 CCE1               jsr semaP
EE75| A0 0C                 ldy #laData+ddData+atTID+2  ; setup tempts for fndTT;  src TID
EE77| B1 14                 lda @rtP,y
>>>>PAGE -  67  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


EE79| 85 00                 sta t0
EE7B| C8                   iny
EE7C| B1 14                 lda @rtP,y
EE7E| 85 01                 sta t1
EE80| A0 03                 ldy #laSrc+2
EE82| B1 14                 lda @rtP,y
EE84| 85 02                 sta t2          ; src node
EE86| A0 08                 ldy #laData+ddSsskt+2
EE88| B1 14                 lda @rtP,y
EE8A| 85 03                 sta t3          ; src socket
EE8C| 64 04                 stz t4          ; src net
EE8E|
EE8E| 64 05                 stz t5
EE90|
EE90| A0 0B                 ldy #laData+ddData+atBtMap+2    ; setup common code
EE92| B1 14                 lda @rtP,y          ; bitMap (seq nbr)
EE94| 85 06                 sta t6
EE96| 88                   dey
EE97| A9 10                 lda #laData+ddData+atData
EE99| 80**                 bra RXATP
EE9B|
EE9B|                   ;*****************************************************************************
EE9B|                   ;*  RXREL, recv'd a TRel.  If we have a SendResponse underway, finish
EE9B|                   ;* if off..  Otherwise, send packet to bit-bucket.
EE9B|                   ;*****************************************************************************
EE9B|                   RXREL
EE9B| A9 20                 lda #SENDRSP        ; any posted?
EE9D| 20 ****               jsr fndTT
EEA0| F0**                 beq $9          ; if not, drop on floor
EEA2|
EEA2|                   ;** we have one, finish it off.
EEA2| BD 0002               lda ttState,x       ; is this one Q'd?
EEA5| C9 21                 cmp #SENDRSPq
EEA7| D0**                 bne $1          ; nope
EEA9|
EEA9|                   ;** this entry is queued for Tx, so handle specially
EEA9| A9 24                 lda #SENDRSPa       ; set for abort
EEAB| 9D 0002               sta ttState,x
EEAE| 86 E9                 stx lstATP          ; signal ATPtask
EEB0| 80**                 bra $9          ;  and, leave
EEB2|
EEB2|                   ;** a SENDRSP is not queued, we can immediately cancel it.
EEA7* 09
EEB2| 9E 0002            $1  stz ttState,x       ; show "not active"
EEB5| BD 0102               lda ttRID,x         ; setup for postSTS
EEB8| A0 00                 ldy #0          ; simply show done
EEBA| 20 23E3               jsr postSTS
EEBD|
EEB0* 0B
EEA0* 1B
EEBD| A2 7C               $9  ldx #ATPlock        ; free up table
EEBF| 20 98E1               jsr semaV
EEC2| 4C CDE4               jmp RCVtask         ;  and, leave
EEC5|
EEC5|                   ;*****************************************************************************
>>>>PAGE -  68  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


EEC5|
EEC5|                   ;*  RXATP, common code to handle received packet.
EEC5|                   ;*****************************************************************************
EE99* 2A
EE6E* 55
EEC5|                   RXATP                   ;** common handler from above
EEC5| 85 07                 sta t7          ; save header length
EEC7| B1 14                 lda @rtP,y          ; fetch atCCI
EEC9| 85 08                 sta t8          ;  save CCI
EECB| 29 C0                 and #atCmsk         ; mask them
EECD| C9 40                 cmp #atREQ          ; TReq?
EECF| F0**                 beq RXREQ
EED1| C9 C0                 cmp #atREL          ; TRel?
EED3| F0C6                 beq RXREL
EED5| C9 80                 cmp #atRSP          ; TResp?
EED7| D0**                 bne $69
EED9| 4C ****               jmp RXRSP
EEDC|
EED7* 03
EEDC| A2 7C             $69 ldx #ATPlock        ; free table if bad
EEDE| 20 98E1               jsr semaV
EEE1| 4C CDE4               jmp RCVtask         ; none of above; ignore frame
```

```
EEE4|
EEE4|                   ;*************************************************************************
EEE4|                   ;*  RXREQ, recv'd a TReq.  If we don't already have the response,
EEE4|                   ;* pass this packet to Host.  Otherwise, send the data..
EEE4|                   ;*************************************************************************
EECF* 13
EEE4|                   RXREQ
EEE4| A9 20                 lda #SENDRSP        ; look for a Response outstanding
EEE6| 20 ****             jsr fndTT           ; find a matching entry
EEE9| F0**                 beq nRXREQ          ; no, must be new one
EEEB|
EEEB|                   ;** we have found our request; check on duplicates
EEEB| BD 0002             lda ttState,x       ; is this new one (waiting)?
EEEE| C9 28               cmp #SENDRSPn
EEF0| F0**                 beq $9          ; yes, ignore this one
EEF2|
EEF2|
EEF2| A5 06               lda t6          ; else, get this entry's TReq BtMap
EEF4| 3D 0502             and ttRsMap,x       ; prevent errors (in lengths)
EEF7| F0**                 beq RXSTS           ; assume response to STS
EEF9| 9D 0602             sta ttTxMap,x       ; update desired ones
EEFC| BD 0002             lda ttState,x       ; what's it doing?
EEFF| C9 23               cmp #SENDRSPw       ; waiting?
EF01| D0**                 bne $9          ;  no, it'll find out
EF03| A9 20               lda #SENDRSP        ; yes, change to do a send
EF05| 9D 0002             sta ttState,x
EF08| 86 E9               stx lstATP          ; show state changed
EF0A|
EF01* 07
EEF0* 18
EF0A| A2 7C             $9  ldx #ATPlock        ; unlock ATP table
EF0C| 20 98E1               jsr semaV
EF0F| 4C CDE4               jmp RCVtask         ;  and, leave
>>>>PAGE -  69  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


EF12|
EF12|                   ;*************************************************************************
EF12|                   ;*  RXSTS, assumed response to a STS (or timeout), where we have sent
EF12|                   ;* all of the packets that Host originally gave us in SendResponse.  Pass
EF12|                   ;* back the info to Host;  it will presumably give us an AddResponse.
EF12|                   ;*************************************************************************
EEF7* 19
EF12| BD 0002           RXSTS   lda ttState,x       ; if SendResp is waiting
EF15| C9 23                 cmp #SENDRSPw
EF17| D0**                 bne $1
EF19| A9 30               lda #30         ;  then, reset timer
EF1B| 9D 0302             sta ttTmr,x
EF1E|
EF17* 05
EF1E| A5 06             $1  lda t6          ; get new map
EF20| 48                   pha             ;  and save it
EF21| DA                   phx             ;  and, x
EF22| A2 74                 ldx #DMAlock        ; but, we do need DMA
EF24| 20 CCE1
EF27| A9 C0                 lda #DMAinfo
EF29| 20 24E4               jsr waitDMA         ; wait for it
EF2C|
EF2C| FA                   plx             ; get entry again
EF2D| A9 74                 lda #074            ; show what kind
EF2F| 8D 0080               sta DMAreg
EF32| BD 0102               lda ttRID,x         ;  and, that it's in process
EF35| 8D 0080               sta DMAreg
EF38| 68                   pla             ; reload the map
EF39| 8D 0080               sta DMAreg
EF3C|
EF3C| 20 3BE4               jsr freeDMA         ; ok, free stuff up
EF3F| A2 7C                 ldx #ATPlock        ; unlock ATP table
EF41| 20 98E1               jsr semaV
EF44| 4C CDE4               jmp RCVtask         ;  and, leave
EF47|
EF47|                   ;*************************************************************************
EF47|                   ;** come here for new request;  note: we do not post a new SENDRSP;  wait
EF47|                   ;** for our Host to ask us to.
EF47|                   ;*  We used to generate table on XO, but now we pass thru (as in olden
EF47|                   ;* times);  Host will handle filtering of duplicates until SENDRSP is issued.
EF47|                   ;*************************************************************************
EEE9* 5C
EF47|                   nRXREQ
EF47|                   ;***    the commented code can be used to do auto-generation of table
EF47|                   ;   lda t8      ; retrieve CCI
EF47|                   ;   and #atXO       ; XO flag set?
EF47|                   ;   beq $8      ;  no, normal one
EF47|
EF47|                   ;****   its an XO;  build a new table entry (if we have room)
EF47|                   ;   ldx lstTTP      ; start from last one allocated.
EF47|                   ;$1 lda ttState,x   ; this one busy?
EF47|                   ;   beq $3      ;  no, we have one
```

```
EF47|
EF47|                   ;   txa         ; yes, advance thru table
EF47|                   ;   clc
>>>>PAGE -  70  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


EF47|                   ;   adc #TTSIZE     ;  by incrementing by table entry size
EF47|                   ;   bne $2      ; skip if not at page boundary
EF47|                   ;   lda #TTBASE % 0100  ; else, start at front
EF47|                   ;$2 tax
EF47|                   ;   cmp lstTTP      ; have we gone around all the way?
EF47|                   ;   bne $1      ;  no, keep looking
EF47|                   ;   bra $69     ; else, we have no room, ignore it..
EF47|
EF47|                   ;** we have an empty entry, X marks the spot; add data for fndTT later
EF47|                   ;$3 stx lstTTP      ; save ptr for next time
EF47|                   ;   lda #SENDRSPn   ; show we're waiting
EF47|                   ;   sta ttState,x   ; save correct NEW state
EF47|                   ;   txa         ; makeup RID
EF47|                   ;   ora #080
EF47|                   ;   sta ttRID,x     ; save for later match on SENDRSP
EF47|                   ;   pha         ;  and, for later
EF47|                   ;   lda #atXO       ; save XO flag
EF47|                   ;   sta ttFLGs,x
EF47|                   ;   lda t4      ; copy for dup checks
EF47|                   ;   sta ttDnet,x
EF47|                   ;   lda t5
EF47|                   ;   sta ttDnet+1,x
EF47|                   ;   lda t2
EF47|                   ;   sta ttDnode,x
EF47|                   ;   lda t3
EF47|                   ;   sta ttDskt,x
EF47|                   ;   lda t0
EF47|                   ;   sta ttTID,x
EF47|                   ;   lda t1
EF47|                   ;   sta ttTID+1,x
EF47|                   ;   lda #30     ; set default timer
EF47|                   ;   sta ttTmr,x
EF47|                   ;   plx         ; restore RID
EF47|                   ;   bra $9      ;  and, merge w/ common code
EF47|
EF47| A2 00             $8  ldx #0      ; no RID (non-XO)
EF49|
EF49| A9 34             $9  lda #034        ; our function code
EF4B| A4 07                 ldy t7      ; pickup header length
EF4D|
EF4D| 20 D5EA               jsr RXDDP1      ;  use shared code to copy
EF50|
EF50| A2 7C             $69 ldx #ATPlock    ; unlock table
EF52| 20 98E1               jsr semaV
EF55| 4C CAE4               jmp RCVfree     ; and, merge
EF58|
EF58|                   ;*************************************************************************
EF58|                   ;*  RXRSP, recv'd a TResp.  If we have the SendRequest under way,
EF58|                   ;* copy the packet to Host (assuming filtering for duplicates, etc.).  If
EF58|                   ;* we do not have SendRequest pending, throw this guy away.
EF58|                   ;*************************************************************************
EEDA* 58EF
EF58|                   RXRSP
EF58| A9 10                 lda #SENDREQ        ; look for a request
EF5A| 20 ****               jsr fndTT
>>>>PAGE -  71  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


EF5D| F0**                  beq $9          ; if none, ignore packet
EF5F|
EF5F|                   ;** we have a TTtbl entry;  check if duplicate.
EF5F| A4 06                 ldy t6          ; pickup TResp's SqNbr
EF61| B9 5CE0               lda atBits,y
EF64| 3C 0603               bit ttBtMap,x       ; this one expected?
EF67| F0**                  beq $9          ;  no, ignore it then
EF69| 5D 0603               eor ttBtMap,x       ; yes, mark off list
EF6C| 9D 0603               sta ttBtMap,x
EF6F|
EF6F|                   ;** process the EOM bit in this TResp packet.
EF6F| A0 0A                 ldy #laData+ddData+atCCI+2  ; assume DDP pkt
EF71| A5 07                 lda t7          ; check on pkt type
EF73| C9 10                 cmp #laData+ddData+atData   ; correct?
EF75| F0**                  beq $1
EF77| A0 12                 ldy #laData+xdData+atCCI+2  ; else, DDPX
EF79|
EF75* 02
EF79| B1 14             $1  lda @rtP,y          ; pickup CCI byte
EF7B| 85 00                 sta t0          ; save
EF7D|
EF7D| 29 10                 and #atEOM          ; is it last one?
EF7F| F0**                  beq $2          ;  nope, keep going
EF81| A4 06                 ldy t6          ; else, clear rest of ttBtMap
```

```
EF83| C8                    iny
EF84| B9 53E0               lda atMaps,y
EF87| 3D 0603               and ttBtMap,x
EF8A| 9D 0603               sta ttBtMap,x
EF8D|
EF8D|                     ;** we have a packet for Host, use common code to pass to Host.
EF7F* 0C
EF8D| 86 E9               $2  stx lstATP          ; save our index
EF8F| A5 00                   lda t0      ; save across task switches
EF91| 48                     pha
EF92| BD 0102               lda ttRID,x         ; setup call to pass it along
EF95| AA                     tax
EF96| A9 36                 lda #036            ;  to Host
EF98| A4 07                 ldy t7
EF9A| 20 D5EA               jsr RXDDP1          ; share code to transfer to Host
EF9D| 68                     pla
EF9E| 85 00                 sta t0
EFA0| A6 E9                 ldx lstATP          ; restore index
EFA2| BD 0603               lda ttBtMap,x       ; any more?
EFA5| F0**                  beq $3          ;  nope, check on XO
EFA7|
EFA7|                     ;** for normal processing, check if STS is on in request.  If so, change state
EFA7|                     ;* to send the request again.
EFA7| A5 00                   lda t0          ; CCI from recv'd pkt
EFA9| 29 08                 and #atSTS          ; STS requested?
EFAB| F0**                  beq $8          ;  no, normal processing
EFAD| BD 0002               lda ttState,x       ; is it already queued?
EFB0| C9 11                 cmp #SENDREQq
EFB2| F0**                  beq $8          ; yes, don't need to play
EFB4| A9 10                 lda #SENDREQ        ; else, set to re-issue the request
EFB6| 80**                  bra $6          ;  and, merge
>>>>PAGE -  72  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


EFB8|
EFB8|                     ;** following is placed here for addressability
EF67* 4F
EF5D* 59
EFB8| A2 7C               $9  ldx #ATPlock        ; free table
EFBA| 20 98E1                 jsr semaV
EFBD| 4C CDE4                 jmp RCVtask         ; toss un-expected one
EFC0|
EFC0|
EFC0|                     ;** whoopee!! the SendRequest is done. finish us off
EFA5* 19
EFC0| BD 0402               $3  lda ttFLGs,x        ; is this XO?
EFC3| 29 20                   and #atXO
EFC5| D0**                  bne $4          ; yes, special processing
EFC7|
EFC7| 9E 0002                 stz ttState,x       ; else, show done immediately
EFCA| BD 0102                 lda ttRID,x         ; set status
EFCD| A0 00                   ldy #0
EFCF| 20 23E3                 jsr postSTS
EFD2| 80**                    bra $8
EFD4|
EFD4|                     ;** XO SendRequest;  send TRel first.
EFC5* 0D
EFD4| BD 0002               $4  lda ttState,x       ; what's it doing?
EFD7| A0 30                   ldy #SENDREL        ; assume we can do it now..
EFD9| C9 11                   cmp #SENDREQq       ; Q'd?
EFDB| D0**                    bne $5          ;  no, set for normal
EFDD| A0 33                   ldy #SENDRELr       ; if Q'd, set for abort
EFDF|
EFDB* 02
EFDF| 98                   $5  tya             ; get appropriate state
EFB6* 28
EFE0| 9D 0002              $6  sta ttState,x
EFE3|
EFD2* 0F
EFB2* 2F
EFAB* 36
EFE3| A2 7C               $8  ldx #ATPlock        ; free table
EFE5| 20 98E1                 jsr semaV
EFE8| 4C CAE4                 jmp RCVfree         ; this one is done...
EFEB|
EFEB|                     ;*************************************************************************
EFEB|                     ;*  fndTT, common routine to find a matching entry in TTtable, based
EFEB|                     ;* upon params in t0..t6 from a recv'd packet.  A has proto-type of desired
EFEB|                     ;* entry type (SENDREQ/SENDRSP).  Upon exit, Z -> entry not found, else X
EFEB|                     ;* has table's index for further amusement and pleasure...
EFEB|                     ;*  Note, the order in which we look at stuff should match expected
EFEB|                     ;* degree of mismatch.
EFEB|
EFEB|                     ;*************************************************************************
EF5B* EBEF
EEE7* EBEF
EE9E* EBEF
EFEB|                     fndTT                   ;** called by RXREQ/RXRSP/RXREL
```

```
EFEB| 85 09                sta t9          ; save for later in loop
>>>>PAGE -  73  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


EFED| A2 09                ldx #TTBASE % 0100      ; start of table
EFEF|
EFEF| BD 0002           $1  lda ttState,x       ; kind of entry
EFF2| 29 F0                and #0F0
EFF4| C5 09                cmp t9          ; this right kind?
EFF6| D0**                 bne $2
EFF8| BD 0803              lda ttTID+1,x       ; OK, look at rest
EFFB| C5 01                cmp t0+1
EFFD| D0**                 bne $2
EFFF| BD 0703              lda ttTID,x
F002| C5 00                cmp t0
F004| D0**                 bne $2
F006| BD 0203              lda ttDnode,x
F009| C5 02                cmp t2
F00B| D0**                 bne $2
F00D| BD 0303              lda ttDskt,x
F010| C5 03                cmp t3
F012| D0**                 bne $2
F014| BD 0003              lda ttDnet,x
F017| C5 04                cmp t4
F019| D0**                 bne $3          ; special for nets
F01B| BD 0103              lda ttDnet+1,x
F01E| C5 05                cmp t5
F020| F0**                 beq $9
F022|
F022|                    ;** net matching is messier than simple compares!!
F019* 07
F022| A5 0C              $3  lda ThisNet         ; if ThisNet is zero
F024| 05 0D                ora ThisNet+1
F026| F0**                 beq $9          ;  accept it
F028| BD 0003              lda ttDnet,x        ; else, if Dnet is zero
F02B| 1D 0103              ora ttDnet+1,x
F02E| D0**                 bne $2          ;  accept it
F030|
F030|                    ;** this looks like a match, return to caller w/ not-Z
F026* 08
F020* 0E
F030| 8A                $9  txa
F031| 60                    rts
F032|
F02E* 02
F012* 1E
F00B* 25
F004* 2C
EFFD* 33
EFF6* 3A
F032| 8A                $2  txa             ; not match, advance ptr
F033| 18                    clc
F034| 69 0D                adc #TTSIZE
F036| AA                    tax
F037| D0B6                  bne $1
F039| 60                    rts             ; Z set if exit this way
F03A|
F03A|                    ;*************************************************************************
F03A|                    ;*  xABTREQ/xABTRSP, abort a possible SendRequest in process.
>>>>PAGE -  74  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F03A|                    ;* The RID param is that of the desired SendRequest.  FC = $39/$3A
F03A|
F03A|                    ;*  We also have one for SNDNBP;  FC= $2F
F03A|                    ;*************************************************************************
F03A| A9 40             xABTNBP lda #SENDNBP        ; add abort for NBP
F03C| 80**                   bra ABTRQRS
F03E|
F03E| A9 10             xABTREQ lda #SENDREQ        ; what we want
F040| 80**                   bra ABTRQRS
F042|
F042| A9 20             xABTRSP lda #SENDRSP
F044|
F040* 02
F03C* 06
F044|                    ABTRQRS                 ;** common code!!
F044| 48                    pha
F045| A2 74                ldx #DMAlock        ; we don't need DMA anymore
F047| 20 98E1              jsr semaV
F04A| A2 7C                ldx #ATPlock        ; make sure we can diddle
F04C| 20 CCE1              jsr semaP
F04F| 68                    pla
F050| 85 00                sta t0          ; save REQ/RSP code
F052| 18                    clc
F053| 69 01                adc #SENDREQq-SENDREQ   ; adjust values
F055| 85 01                sta t1
F057| 69 03                adc #SENDREQa-SENDREQq
```

```
F059| 85 02                sta t2
F05B|
F05B| A2 09                ldx #TTBASE % 0100
F05D| A5 47             $0  lda CMDRID          ; the one we're looking for
F05F|
F05F| DD 0102           $1  cmp ttRID,x         ; this one?
F062| D0**                  bne $8
F064| BD 0002                lda ttState,x       ; make sure
F067| C5 01                  cmp t1          ; is it queued entry?
F069| D0**                  bne $2
F06B| A5 02                  lda t2          ; yes, show to be aborted
F06D| 9D 0002                sta ttState,x
F070| 86 E9                  stx lstATP          ; show one changed
F072| 64 47                  stz CMDRID          ; clear RID to skip status
F074| 80**                  bra $9          ;  and, leave
F076|
F069* 0B
F076| 29 F0             $2  and #0F0            ; check if its any SENDREQ
F078| C5 00                  cmp t0
F07A| D0**                  bne $69
F07C| C9 40                  cmp #SENDNBP        ; special?
F07E| D0**                  bne $3          ; no
F080| DA                    phx             ; save X across call
F081| BD 0402                lda ttBufN,x        ;  yes, free buffer used
F084| 20 73E2                jsr putFree         ;   by SNDNBP
F087|
F087| FA                    plx
F088| 64 CC                  stz nbTTP           ; show done..
F08A|
F07E* 0A
>>>>PAGE -  75  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F08A| 9E 0002           $3  stz ttState,x       ; show finished
F08D| A9 02                  lda #ABTdone        ;  and, its outcome
F08F| 85 45                  sta CMDRslt
F091| 80**                  bra $9
F093|
F062* 2F
F093| 8A                $8  txa             ; crank to next
F094| 18                    clc
F095| 69 0D                  adc #TTSIZE
F097| D0C6                  bne $1
F099|
F07A* 1D
F099| A9 FF            $69 lda #REQerr         ; show error
F09B| 85 45                  sta CMDRslt
F09D|
F091* 0A
F074* 27
F09D| A2 7C             $9  ldx #ATPlock        ; free-up table
F09F| 20 98E1                jsr semaV
F0A2| 4C 4CE3                jmp CMDstat
F0A5|
F0A5|                   ;*************************************************************************
F0A5|                   ;*  xSNDRSP, the entry for a Host ATPSendResponse.  Note that the proto-
F0A5|                   ;* type ATP packet header has been set by Host software.  This entry is called
F0A5|                   ;* by CMDtask, and returns to it.  However, the actual transmission is done
F0A5|                   ;* by ATPtask.  FC = $32.
F0A5|                   ;*  Before building a new entry, we scan in case this is a response to an
F0A5|                   ;* XO TReq.  If so, we need not allocate a new one, just use an old one.
F0A5|                   ;*************************************************************************
F0A5|                   xSNDRSP         ;** entry for SendResponse;  runs as CMDtask
F0A5| A5 47                  lda CMDRID      ; get Host's value
F0A7| 10**                  bpl $9      ; if not one of our's, assume non-XO
F0A9|
F0A9| A6 E8                  ldx lstTTP      ; start from last one allocated.
F0AB| BD 0002           $1  lda ttState,x   ; this one busy?
F0AE|
F0AE| C9 28                  cmp #SENDRSPn   ; waiting one?
F0B0| D0**                  bne $2
F0B2| BD 0102                lda ttRID,x     ; RID match?
F0B5| C5 47                  cmp CMDRID
F0B7| D0**                  bne $2
F0B9| A9 20                  lda #SENDRSP    ; yes, re-use this entry for our response
F0BB| 48                    pha
F0BC| 80**                  bra bldTT1      ;  and, merge
F0BE|
F0B7* 05
F0B0* 0C
F0BE| 8A                $2  txa         ; yes, advance thru table
F0BF| 18                    clc
F0C0| 69 0D                  adc #TTSIZE     ;  by incrementing by table entry size
F0C2| D0**                  bne $3      ; skip if not at page boundary
F0C4| A9 09                  lda #TTBASE % 0100  ; else, start at front
F0C2* 02
F0C6| AA                $3  tax
>>>>PAGE -  76  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk
```

```
F0C7| C5 E8                 cmp lstTTP      ; have we gone around all the way?
F0C9| D0E0                 bne $1      ;  no, keep looking
F0CB|
F0CB|                    ;** if no match, assume normal entry
F0A7* 22
F0CB| A9 20              $9  lda #SENDRSP    ; set state value
F0CD| 80**                 bra bldTT       ;  and, share common entry
F0CF|
F0CF|                    ;*************************************************************************
F0CF|                    ;*  xSNDREQ, the entry for a Host ATPSendRequest.  Note that the proto-
F0CF|                    ;* type ATP packet header has been set by Host software.  This entry is called
F0CF|                    ;* by CMDtask, and returns to it.  However, the actual transmission is done
F0CF|                    ;* by ATPtask.  FC = $31
F0CF|                    ;*************************************************************************
F0CF|                    xSNDREQ         ;** entry for SendRequest;  runs as CMDtask
F0CF| A9 10                 lda #SENDREQ    ; set state value
F0D1|
F0D1|
F0D1|                    ;*************************************************************************
F0D1|                    ;*  bldTT, common code to build a new TT entry;  the format for both
F0D1|                    ;* SNDREQ and SNDRSP is identical.
F0D1|                    ;*************************************************************************
F0CD* 02
F0D1|                    bldTT       ;** common entry
F0D1| 48                     pha         ; save tState value
F0D2|
F0D2| A6 E8                 ldx lstTTP      ; start from last one allocated.
F0D4| BD 0002            $1  lda ttState,x   ; this one busy?
F0D7| F0**                 beq $3      ;  no, we have one
F0D9| 8A                   txa         ; yes, advance thru table
F0DA| 18                   clc
F0DB| 69 0D                adc #TTSIZE     ;  by incrementing by table entry size
F0DD| D0**                 bne $2      ; skip if not at page boundary
F0DF| A9 09                lda #TTBASE % 0100  ; else, start at front
F0DD* 02
F0E1| AA                 $2  tax
F0E2| C5 E8                 cmp lstTTP      ; have we gone around all the way?
F0E4| D0EE                 bne $1      ;  no, keep looking
F0E6|
F0E6|                    ;** we haven't found a free one; signal error to CMD caller.
F0E6| 68                   pla         ; pop saved state
F0E7| A9 FE                lda #NoFreeErr
F0E9| 4C 45E3              jmp CMDdone     ;  and, thats it
F0EC|
F0EC|                    ;** we have an empty entry, X marks the spot
F0D7* 13
F0EC| 9E 0402            $3  stz ttFLGs,x    ; fixup this for non-X0
F0EF|
F0BC* 31
F0EF| 86 E8              bldTT1  stx lstTTP      ; save ptr for next time
F0F1| 68                   pla         ; restore desired state
F0F2| 9D 0002              sta ttState,x   ; save correct NEW state
F0F5|
F0F5|                    ;** now, build rest of entries from DMA
F0F5|
>>>>PAGE -  77  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F0F5| A5 47                 lda CMDRID      ; copy RID
F0F7| 9D 0102               sta ttRID,x
F0FA| DA                   phx         ; save X across loop
F0FB|
F0FB|                    ;** copy rest of request using admittedly sleazy code
F0FB| A0 09                 ldy #ttUser-ttDnet  ; loop counter
F0FD| AD 0080            $1  lda DMAreg
F100| 9D 0003              sta ttDnet,x    ; right on for first one
F103| E8                   inx
F104| 88                   dey
F105| D0F6                 bne $1
F107|
F107| FA                   plx         ; pickup our ptr
F108| 86 E9                stx lstATP      ; signal ATPtask
F10A| BD 0402              lda ttFLGs,x    ; keep old X0 flag
F10D| 29 20                and #atX0
F10F| 1D 0503              ora ttCCI,x     ; add in our flags
F112| 9D 0402              sta ttFLGs,x
F115| BD 0002              lda ttState,x   ; what is this?
F118| C9 10                cmp #SENDREQ    ; SendRequest?
F11A| F0**                 beq bldTREQ
F11C|
F11C|                    ;** finish setup for SNDRSP
F11C| BD 0402            bldTRSP lda ttFLGs,x    ; set initial value
F11F| 29 20                and #atX0       ;  (passing X0)
F121| 09 80                ora #atRSP
F123| 9D 0503              sta ttCCI,x
F126| BD 0603              lda ttBtMap,x   ; then, # of buffers for SendResponse
```

```
F129| 9D 0602              sta ttTxMap,x   ; save it for later
F12C| 9D 0502              sta ttRsMap,x   ;  (copy for EOM checks)
F12F|
F12F| 64 47            $9  stz CMDRID      ; show no status now
F131| 4C 45E3              jmp CMDdone     ;  and, wait for more
F134|
F134|                  ;** finish setup actions for SNDREQ
F11A* 18
F134| AD 0080          bldTREQ lda DMAreg      ; copy User bytes
F137| 9D 0903              sta ttUser,x
F13A| AD 0080              lda DMAreg
F13D| 9D 0A03              sta ttUser+1,x
F140| AD 0080              lda DMAreg
F143| 9D 0B03              sta ttUser+2,x
F146| AD 0080              lda DMAreg
F149| 9D 0C03              sta ttUser+3,x
F14C|
F14C| AD 0080              lda DMAreg      ; copy control stuff
F14F| 9D 0502              sta ttTmOut,x
F152| AD 0080              lda DMAreg
F155| 9D 0602              sta ttRtrys,x
F158| AD 0080
F15B| 9D 0702              sta ttDlng,x
F15E| AD 0080              lda DMAreg
F161| 9D 0802              sta ttDlng+1,x
F164|
F164| BD 0402              lda ttFLGs,x    ; mask bits for TReq
>>>>PAGE -  78  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F167| 29 20                and #atXO
F169| 09 40                ora #atREQ      ; add in request bits
F16B| 9D 0503              sta ttCCI,x
F16E|
F16E| BC 0603          $1  ldy ttBtMap,x   ; convert # of responses to
F171| B9 53E0              lda atMaps,y    ;  bitMap
F174| 9D 0603              sta ttBtMap,x
F177|
F177| 64 47            $9  stz CMDRID      ; show no status now
F179| 4C 45E3              jmp CMDdone     ;  and, wait for more
F17C|
F17C|                  ;*****************************************************************************
F17C|                  ;*  xADDRSP, code to support AddResponse function.  We search the TTtbl
F17C|                  ;* for active entry that matches our RID.  If we find an outstanding SENDRSP,
F17C|                  ;* we update its ttRsMap with new values.
F17C|                  ;*****************************************************************************
F17C|                  xADDRSP             ;** entry from CMDTASK
F17C| AD 0080              lda DMAreg      ; get CCI/FLGS
F17F| 29 18                and #atEOM+atSTS    ; must have one of these
F181| F0**                beq $69
F183| 85 00                sta t0      ; else, save
F185| AD 0080              lda DMAreg      ; get map
F188| F0**                beq $69
F18A| 85 01                sta t1
F18C| A2 09                ldx #TTBASE % 0100  ; start looking
F18E|
F18E| BD 0002          $1  lda ttState,x   ; must be SENDRSP type
F191| 29 F0                and #0F0
F193| C9 20                cmp #SENDRSP
F195| D0**                bne $2      ; if not, keep looking
F197| BD 0102              lda ttRID,x     ; else, match?
F19A| C5 47                cmp CMDRID
F19C| F0**                beq $5      ;  yes, further look
F19E|
F195* 07
F19E| 8A                $2  txa         ; else, advance search
F19F|
F19F| 18                    clc
F1A0| 69 0D                adc #TTSIZE
F1A2| AA                    tax
F1A3| D0E9                bne $1
F1A5|
F1A5| A9 FC                lda #NoTTErr    ; we looked, and no match
F1A7| 4C 45E3              jmp CMDdone     ;  so, report error
F1AA|
F188* 20
F181* 27
F1AA| A9 FF            $69 lda #REQerr     ; show error
F1AC| 4C 45E3              jmp CMDdone
F1AF|
F1AF|                  ;** we have what is presumed to be our entry of interest
F19C* 11
F1AF| A5 00            $5  lda t0      ; update stuff
F1B1| 9D 0402              sta ttFLGs,x
F1B4| A5 01                lda t1
>>>>PAGE -  79  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk
```

```
F1B6| 9D 0502               sta ttRsMap,x
F1B9| 9D 0602               sta ttTxMap,x
F1BC| A9 20                 lda #SENDRSP    ; setup for ATPtask
F1BE| 9D 0002               sta ttState,x
F1C1| 86 E9                 stx lstATP      ; force ATPscan to go active
F1C3| 64 47                 stz CMDRID      ; show no error status
F1C5| 4C 45E3               jmp CMDdone     ;  and, merge
F1C8|
F1C8|                    ;*******************************************************************************
F1C8|                    ;*  bldHdr, build an ATP header in atPh area from TT entry whose offset
F1C8|                    ;* is currently in X.
F1C8|                    ;*******************************************************************************
F1C8|                    bldHdr              ; used by NEWREQ and NXTRSP
F1C8| BD 0003               lda ttDnet,x    ; check if its long packet stuff
F1CB| 1D 0103               ora ttDnet+1,x
F1CE| F0**                  beq bldDDP      ; do shorty
F1D0|
F1D0|                    ;** check on which net to use
F1D0| A5 0B                 lda aBridge     ; valid bridge?
F1D2| D0**                  bne $1      ;  yes, ThisNet is valid!
F1D4| A5 0C                 lda ThisNet     ; else, is ThisNet also 0?
F1D6| 05 0D                 ora ThisNet+1
F1D8| F0**                  beq bldDDPX     ; if yes, force DDPX
F1DA|
F1D2* 06
F1DA| BD 0003            $1  lda ttDnet,x    ; if to a net, is it ours?
F1DD|
F1DD| C5 0C                 cmp ThisNet
F1DF| D0**                  bne bldDDPX     ; nope, must be long
F1E1| BD 0103               lda ttDnet+1,x
F1E4| C5 0D                 cmp ThisNet+1
F1E6| F0**                  beq bldDDP      ; yeah, we can use short one
F1E8|
F1E8|                    ;** bldDDPX, build an eXtended form of DDP
F1DF* 07
F1D8* 0E
F1E8| A5 0B              bldDDPX lda aBridge     ; send to a bridge on our net
F1EA| D0**                  bne $1      ; if valid, use it
F1EC| BD 0203               lda ttDnode,x   ;  else, use caller's
F1EF|
F1EA* 03
F1EF| A0 00              $1  ldy #laDst          ; pt to Dst
F1F1| 91 E2                 sta @atPh,y
F1F3| C8                    iny
F1F4| A5 69                 lda myNode
F1F6| 91 E2                 sta @atPh,y
F1F8| C8                    iny
F1F9| A9 02                 lda #laDDPX
F1FB| 91 E2                 sta @atPh,y
F1FD| C8                    iny
F1FE|
F1FE| 18                    clc             ; compute & stash count
F1FF| A5 E4                 lda atC         ; this is length of ATP data
F201| 69 15                 adc #xdData+atData      ;  + headers
F203| 48                    pha             ; reverse order!
>>>>PAGE -  80  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F204| A5 E5                 lda atC+1
F206| 69 00                 adc #0
F208| 91 E2                 sta @atPh,y
F20A| C8                    iny
F20B| 68                    pla
F20C| 91 E2                 sta @atPh,y
F20E| C8                    iny
F20F|
F20F|                    ;***    B4 change, added CKS, sNode to header
F20F| A9 00                 lda #0
F211| 91 E2                 sta @atPh,y         ; CKS
F213| C8                    iny
F214| 91 E2                 sta @atPh,y
F216| C8                    iny
F217| BD 0003               lda ttDnet,x
F21A| 91 E2                 sta @atPh,y
F21C| C8                    iny
F21D| BD 0103               lda ttDnet+1,x
F220| 91 E2                 sta @atPh,y
F222| C8                    iny
F223| A5 0C                 lda ThisNet
F225| 91 E2                 sta @atPh,y
F227| C8                    iny
F228| A5 0D                 lda ThisNet+1
F22A| 91 E2                 sta @atPh,y
F22C| C8                    iny
F22D| BD 0203               lda ttDnode,x
F230| 91 E2                 sta @atPh,y
F232| C8                    iny
F233| B5 69                 lda myNode,x        ; Snode
```

```
F235| 91 E2                sta @atPh,y
F237| C8                   iny
F238| 80**                 bra bldHdr1         ; merge paths
F23A|
F23A|                   ;** build LAP/DDP header (short)
F1E6* 52
F1CE* 6A
F23A| A0 00             bldDDP  ldy #laDst          ; pt to laDst
F23C| BD 0203               lda ttDnode,x
F23F| 91 E2                sta @atPh,y
F241| C8                   iny
F242| A5 69
F244| 91 E2                sta @atPh,y
F246| C8                   iny
F247| A9 01                lda #laDDP
F249| 91 E2                sta @atPh,y
F24B| C8                   iny
F24C| 18                   clc             ; compute & stash count
F24D| A5 E4                lda atC         ; this is length of ATP data
F24F| 69 0D                adc #ddData+atData      ;  + headers
F251| 48                   pha             ; reverse order!
F252| A5 E5                lda atC+1
F254| 69 00                adc #0
F256| 91 E2                sta @atPh,y
F258| C8                   iny
>>>>PAGE -  81  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F259| 68                   pla
F25A| 91 E2                sta @atPh,y
F25C| C8                   iny
F25D|
F25D|                   ;** common path to build rest of DDP/ATP header
F238* 23
F25D| BD 0303             bldHdr1 lda ttDskt,x        ; finish trail end of DDP
F260| 91 E2                sta @atPh,y
F262| C8                   iny
F263| BD 0403               lda ttSskt,x
F266| 91 E2                sta @atPh,y
F268| C8                   iny
F269| A9 03                lda #ddATP
F26B| 91 E2                sta @atPh,y
F26D| C8                   iny
F26E|
F26E|                   ;** finally, the ATP header (copied by sleazy code!)
F26E| DA                   phx             ; save X
F26F| A9 08                lda #atData
F271| 85 00                sta t0          ; loop counter
F273| BD 0503             $1  lda ttCCI,x         ; note! X advances (pretty sneaky!!)
F276| 91 E2                sta @atPh,y
F278| C8                   iny
F279| E8                   inx
F27A| C6 00                dec t0
F27C| D0F5                bne $1
F27E| FA                   plx             ; recover X
F27F|
F27F|                   ;** note! Y now has size of the header
F27F| 98                   tya
F280| 92 E0                sta @atP            ; set length for XMTtask
F282| 60                   rts
F283|
F283|
F283|                   ;*************************************************************************
F283|                   ;*  ATPtask, the actual task which manages all ATP stuff.  SendRequests
F283|                   ;* and/or SendResponses result in a table entry being added;  ATPtask is then
F283|                   ;* Signalled, whereupon it takes the desired action.
F283|                   ;*  The dynamic behaviour of ATP revolves around the ATPloop.  Note that
F283|
F283|                   ;* there are 3 distinct loops in ATPtask, each "guarded" by checking if an
F283|                   ;* event has been noted which requires its action.
F283|                   ;*************************************************************************
F283| A2 E0             ATPtask ldx #atP        ; grab a single buffer (for all time!!)
F285| 20 33E2               jsr getFree
F288| 85 5C                sta ATPBufN     ; save its number for interest
F28A| 18                   clc         ; create header ptr
F28B| A5 E0                lda atP
F28D| 69 68                adc #HDROFS % 0100
F28F| 85 E2                sta atPh
F291| A5 E1                lda atP+1
F293| 69 02                adc #HDROFS / 0100
F295| 85 E3                sta atPh+1
F297| A0 01                ldy #1      ; set ATP as task to wakeup on TXDONE
F299| A9 58                lda #ATPtcb
>>>>PAGE -  82  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F29B| 91 E0                sta @atP,y
F29D|                   ;** from now on, atP/atPh are considered correct;  make sure they are!!
```

```
F29D|
F29D|                   ;*************************************************************************
F29D|                   ;*  ATPloop, the main path for waiting for interesting events.  These
F29D|                   ;* consist of:
F29D|                   ;*  1)  ONESEC, a one-second interval has occurred.
F29D|                   ;*  2)  NEWREQ, a new host request has been processed.
F29D|                   ;*  3)  TXDONE, "THE" atP buffer has been transmitted.
F29D|                   ;*************************************************************************
F29D| 20 F7E1           atpLoop jsr Yield       ; give others a chance
F2A0|
F2A0|                   ;*************************************************************************
F2A0|
F2A0|                   ;*  atpTIME, perform the timout chores related to SendRequests.
F2A0|                   ;*************************************************************************
F2A0| A5 1F             atpTIME lda ATPsecs     ; has time passed?
F2A2| C5 1C                 cmp TMRsecs
F2A4| D0**                  bne $0
F2A6| 4C ****               jmp atpScan     ; if not, just scan
F2A9|
F2A4* 03
F2A9| A2 7C             $0  ldx #ATPlock    ; Yes, lock table for processing
F2AB| 20 CCE1               jsr semaP
F2AE|
F2AE|                   ;** do a quick scan thru table, changing states of interest to timeouts
F2AE| A2 09                 ldx #TTBASE % 0100  ; start at front
F2B0| BD 0002           $1  lda ttState,x   ; get current state
F2B3| F0**                  beq $9      ; skip if null
F2B5|
F2B5| C9 12                 cmp #SENDREQx   ; request sent?
F2B7| F0**                  beq $11     ;  nope, skip this one
F2B9| C9 42                 cmp #SENDNBPx   ; likewise, for SNDNBP
F2BB| D0**                  bne $2
F2BD|
F2B7* 04
F2BD| FE 0002           $11 inc ttState,x   ; yes, start timing (in SENDREQw/SENDNBPw)
F2C0| BD 0502               lda ttTmOut,x
F2C3| 9D 0302               sta ttTmr,x
F2C6| 80**                  bra $9      ;  and, loop
F2C8|
F2BB* 0B
F2C8| C9 13             $2  cmp #SENDREQw   ; timing for real
F2CA| F0**                  beq $21
F2CC| C9 43                 cmp #SENDNBPw   ; share code for SNDNBP
F2CE| D0**                  bne $5
F2D0|
F2CA* 04
F2D0| DE 0302           $21 dec ttTmr,x     ; yes, downcount it
F2D3| D0**                  bne $9
F2D5| BD 0502               lda ttTmOut,x   ; assume more tries ok
F2D8| 9D 0302               sta ttTmr,x
F2DB|
F2DB| BD 0602               lda ttRtrys,x   ; more to go?
F2DE| F0**                  beq $69     ;  no, finish
>>>>PAGE -  83  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F2E0|
F2E0| C9 FF                 cmp #0FF        ; is the retry value $FF
F2E2| F0**                  beq $4      ;  yes, "infinite"
F2E4| 3A                    dea         ; update counter
F2E5| 9D 0602
F2E8|
F2E2* 04
F2E8| BD 0002           $4  lda ttState,x   ; restore original request
F2EB| 29 F0                 and #0F0
F2ED| 9D 0002               sta ttState,x
F2F0| 86 E9                 stx lstATP      ; wakeup atpSCAN
F2F2| 80**                  bra $9
F2F4|
F2CE* 24
F2F4| C9 28             $5  cmp #SENDRSPn   ; new one (not SENDRSP yet?)
F2F6| D0**                  bne $6
F2F8| DE 0302               dec ttTmr,x     ; yes, keep going?
F2FB| D0**                  bne $9      ; yes
F2FD| 9E 0002               stz ttState,x   ;  no, just re-use it
F300| 80**                  bra $9
F302|
F2F6* 0A
F302| C9 23             $6  cmp #SENDRSPw   ; Response waiting?
F304| D0**                  bne $9
F306| DE 0302               dec ttTmr,x     ; yes, down-count it
F309| F0**                  beq $69     ;  abort if no more
F30B|
F304* 05
F300* 09
F2FB* 0E
F2F2* 17
F2D3* 36
```

```
F2C6* 43
F2B3* 56
F30B| 8A                $9  txa         ; advance to next entry
F30C| 18                    clc
F30D| 69 0D                 adc #TTSIZE
F30F| AA                    tax
F310| D09E                  bne $1      ; back for more if not done
F312|
F312| A5 1C                 lda TMRsecs     ; else,
F314| 85 1F                 sta ATPsecs     ;  set for next time
F316| A2 7C                 ldx #ATPlock    ; free table
F318| 20 98E1               jsr semaV
F31B| 80**                  bra atpSCAN
F31D|
F31D|                   ;** xREQFAIL/xRSPFAIL, a SendRequest/SendResponse has timed out
F309* 12
F2DE* 3D
F31D| DA                $69 phx         ; save table entry
F31E| A0 FD                 ldy #TmOutErr   ; signal error
F320| BD 0102               lda ttRID,x     ; show which RID
F323| 20 23E3               jsr postSTS     ;  OK, post status...
F326| FA                    plx         ; get entry ptr back
F327|
>>>>PAGE -  84  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F327| BD 0002               lda ttState,x   ; get state
F32A| 9E 0002               stz ttState,x   ; show free
F32D| 29 F0                 and #0F0        ;  check if NBP
F32F| C9 40                 cmp #SENDNBP
F331| D0D8                  bne $9      ; if not, leave
F333|
F333| DA                    phx         ; save X
F334| BD 0402               lda ttBufN,x    ; free buffer
F337| 20 73E2               jsr putFree
F33A| FA                    plx
F33B| 64 CC                 stz nbTTP       ; then, clear it
F33D| 80CC                  bra $9      ;  and, look for next
F33F|
F33F|
F33F|                   ;*************************************************************************
F33F|                   ;*  atSREQz,atSRSPz,atSRELz  - final processing for ABT's or SENDREL.
F33F|                   ;*************************************************************************
F33F|                   atSNBPz             ; ABT NBP after Q'd
F33F| DA                    phx         ; save X
F340| BD 0402               lda ttBufN,x    ; free buffer
F343| 20 73E2               jsr putFree
F346| FA                    plx
F347| 64 CC                 stz nbTTP       ; show done
F349|
F349|                   atSREQz             ; ABT REQ after Q'd
F349|                   atSRSPz             ; ABT RSP    "   "
F349| A0 02                 ldy #ABTdone    ; final status
F34B| 80**                  bra atSzzzz     ; merge
F34D|
F34D| A0 00             atSRELz ldy #0      ; normal after TRel sent
F34F|
F34B* 02
F34F| DA                atSzzzz phx         ; save X
F350| BD 0102               lda ttRID,x     ; signal Host
F353| 20 23E3               jsr postSTS
F356| FA                    plx
F357| 9E 0002               stz ttState,x   ;  and, show its free
F35A| 80**                  bra atpScn2
F35C|
F35C|                   ;*************************************************************************
F35C|                   ;*  atSREL - set up a TRel.  We simply change the CCI flags to TRel,
F35C|                   ;* zero the request length, and "look like" a SENDREQ (sort of).
F35C|                   ;*************************************************************************
F35C| BD 0503           atSREL  lda ttCCI,x     ; add in FC
F35F| 29 3F                 and #0FF-atCmsk
F361| 09 C0                 ora #atREL      ; set flag
F363| 9D 0503               sta ttCCI,x
F366| 9E 0702               stz ttDlng,x
F369| 9E 0802               stz ttDlng+1,x
F36C| A9 30                 lda #SENDREL    ; set (-1) for later
F36E|
F36E| 9D 0002               sta ttState,x
F371| 80**                  bra atSxxx      ;  then, merge (increments ttState!)
F373|
F373|                   ;*************************************************************************
>>>>PAGE -  85  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F373|                   ;*  atpSCAN, process state changes related to new stuff.
F373|                   ;*************************************************************************
F31B* 56
F2A7* 73F3
```

```
F373| A5 E9             atpSCAN lda lstATP      ; anything of interest?
F375| D0**                 bne $1      ;  yes, start us off
F377| 4C ****             jmp atpXMIT     ; else, advance
F37A|
F375* 03
F37A| A2 7C             $1  ldx #ATPlock    ; get table
F37C| 20 CCE1               jsr semaP
F37F| A2 09                 ldx #TTBASE % 0100  ; else, scan thru table
F381|
F381| BD 0002           atpScn1 lda ttState,x   ; pickup state
F384| F0**                 beq atpScn2     ; skip on none (idle entry)
F386| C9 15                cmp #SENDREQz   ; after abort request
F388| F0BF                beq atSREQz
F38A| C9 45                cmp #SENDNBPz   ; after abort
F38C| F0B1                beq atSNBPz
F38E| C9 25                cmp #SENDRSPz   ; recv'd TRel while Q'd
F390| F0B7                beq atSRSPz
F392| C9 32                cmp #SENDRELz   ; sent TRel, finish SENDREQ
F394| F0B7                beq atSRELz
F396| C9 10                cmp #SENDREQ    ; if SENDREQ or
F398| F0**                beq atSREQ
F39A| C9 20                cmp #SENDRSP    ; else SENDRSP (first time)
F39C| F0**                beq atSRSP
F39E| C9 22                cmp #SENDRSPx   ; else SENDRSP (2nd, and later)
F3A0| F0**                beq atSRSPx
F3A2| C9 30                cmp #SENDREL    ; else SENDREL (normal)
F3A4| F0B6                beq atSREL
F3A6| C9 34                cmp #SENDRELx   ; else SENDREL (after Q'd SENDREQ)
F3A8| F0B2                beq atSREL
F3AA| C9 40                cmp #SENDNBP    ; or SENDNBP (each new send)
F3AC|
F3AC| F0**                beq atSXXX      ;  requires no additional processing
F3AE|
F384* 28
F35A* 52
F3AE| 8A               atpScn2 txa         ; advance stuff
F3AF| 18                   clc
F3B0| 69 0D                adc #TTSIZE
F3B2| AA                   tax
F3B3| D0CC                 bne atpScn1
F3B5|
F3B5| 64 E9                stz lstATP      ; show processed
F3B7| A2 7C                 ldx #ATPlock    ;  and, free table
F3B9| 20 98E1               jsr semaV
F3BC| 4C ****             jmp atpXMIT     ; when scan is complete, on to next stage
F3BF|
F398* 25
F3BF| 9E 0302           atSREQ  stz ttBfr,x     ; save buffer# for atpXMIT
F3C2|
F3C2|                  ;** add this entry to the atTXQ;  note: tricky reg sequences!
F3AC* 14
F371* 4F
>>>>PAGE -  86  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F3C2| FE 0002           atSxxx  inc ttState,x   ; update its state
F3C5| 9E 0202               stz ttLink,x    ; end-of-list
F3C8| 8A                   txa         ; save for later
F3C9| A6 ED                ldx atTXQtl     ; any at end?
F3CB| D0**                 bne $2      ;  yes, update queue
F3CD| 85 EC                sta atTXQhd     ; else, make this only one
F3CF| 80**                 bra $3
F3D1|
F3CB* 04
F3D1| 9D 0202           $2  sta ttLink,x
F3D4|
F3CF* 03
F3D4| AA               $3  tax         ; always add us as last
F3D5| 86 ED                stx atTXQtl
F3D7| 80D5                 bra atpScn2     ; back for more
F3D9|
F3D9|                  ;** figure out next response to send
F3A0* 37
F3D9| A9 20             atSRSPx lda #SENDRSP    ; fix ttState for atSXXX
F3DB| 9D 0002               sta ttState,x
F3DE|
F39C* 40
F3DE| BD 0602           atSRSP  lda ttTxMap,x   ; get current map
F3E1| D0**                 bne $2      ;  if any left, process them
F3E3| BD 0402               lda ttFLGs,x    ; else, check if this is XO (or STS)
F3E6| 29 28                and #atXO+atSTS
F3E8| D0**                 bne $1      ; it is XO, set to wait
F3EA|
F3EA|                  ;** for non-XO, post complete after last response packet out.
F3EA| 9E 0002               stz ttState,x   ;  else, finish this one off
F3ED|
F3ED| DA                   phx         ; save X
F3EE| BD 0102               lda ttRID,x     ; and, post status
```

```
F3F1| A0 00               ldy #0
F3F3| 20 23E3             jsr postSTS
F3F6| FA                  plx         ; recover X
F3F7| 80B5                bra atpScn2
F3F9|
F3F9|                   ;** XO TResp has finished, start timing it
F3E8* 0F
F3F9| A9 23             $1  lda #SENDRSPw   ; set its state
F3FB| 9D 0002             sta ttState,x
F3FE| A9 30               lda #30     ; set default timer
F400| 9D 0302             sta ttTmr,x
F403| 80A9               bra atpScn2
F405|
F405|                   ;** we have at least one more response to go, set it up
F3E1* 22
F405| BD 0503           $2  lda ttCCI,x     ; clear EOM in CCI
F408| 29 E7               and #0FF-atEOM-atSTS
F40A| 9D 0503             sta ttCCI,x
F40D| A0 00               ldy #0      ; look at first remaining bit
F40F| BD 0602             lda ttTxMap,x   ; get BtMap back
F412|
>>>>PAGE -  87  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F412| 4A                $3  lsr A       ;  in map
F413| B0**                 bcs $4
F415| C8                   iny
F416| 80FA                 bra $3
F418|
F413* 03
F418| 98                $4  tya         ; setup the SqNbr
F419| 9D 0603             sta ttSqNbr,x
F41C| 9D 0302             sta ttBfr,x     ; save for atpXMIT
F41F| B9 5CE0             lda atBits,y    ;  and, show sent
F422| 5D 0602             eor ttTxMap,x
F425| 9D 0602             sta ttTxMap,x
F428| D098                 bne atSxxx      ;  if non-zero, more to go
F42A|
F42A|                   ;** we have just set last used
F42A| BD 0402           $5  lda ttFLGs,x    ; else, add EOM/STS flag
F42D| 29 18               and #atEOM+atSTS    ;  (only one s.b. sent)
F42F| 1D 0503             ora ttCCI,x     ; else, add EOM flag;  this is last one
F432| 9D 0503             sta ttCCI,x
F435| 808B               bra atSxxx      ; merge for common code
F437|
F437|                   ;*************************************************************************
F437|                   ;*  atXNBP, send SNDNBP via ATP task.  We get here from below
F437|
F437|                   ;* when atpXMIT recognizes this special send.  Notice that the buffer is not!!
F437|                   ;* the ATP buffer;  rather, the buffer is in ttBufN cell.
F437|                   ;* NOTE: THIS CODE WAS ADDED FOR REV-B
F437|                   ;*************************************************************************
F437| BD 0402           atXNBP  lda ttBufN,x    ; pickup real buffer to use
F43A| 20 80E2             jsr putXmit     ; put into XMT queue
F43D| 4C 9DF2             jmp atpLoop     ;  and, loop.
F440|
F440|                   ;*************************************************************************
F440|                   ;*  atpXMIT, check if any previous TX may have completed (or we don't
F440|                   ;* have one currently).  If so, pull an entry off atTXQ and start it out..
F440|                   ;*************************************************************************
F3BD* 40F4
F378* 40F4
F440| 20 F7E1           atpXMIT jsr Yield       ; give someone a shot
F443|
F443| A6 EE                 ldx xmtATP      ; is one waiting?
F445| F0**                 beq $1      ;  nope, look for one off atTXQ
F447| A9 04               lda #sTXDONE    ; else, is transmit done?
F449| 14 58               trb ATPFlgs
F44B| D0**                 bne $0      ; yes, process it
F44D|
F44D| 4C 9DF2           $9  jmp atpLoop     ;  and, sleep....
F450|
F450|                   ;** a Tx, upon which we have been waiting, has finished.
F44B* 03
F450| FE 0002           $0  inc ttState,x   ; advance state (to SENDXXXx)
F453| 86 E9               stx lstATP      ; signal atpSCAN
F455|
F455| 64 EE                 stz xmtATP      ;  and, show done
F457|
>>>>PAGE -  88  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F457|                   ;** check if any more in atTXQ.
F445* 10
F457| A6 EC             $1  ldx atTXQhd     ; head of queue
F459| F0F2                 beq $9      ;  none, just leave
F45B|
F45B| 86 EE                 stx xmtATP      ; yes, mark as our "current" one
```

```
F45D| BD 0202               lda ttLink,x    ;  and, pull it off Q
F460| 85 EC                 sta atTXQhd
F462| D0**                  bne $2
F464| 64 ED                 stz atTXQtl     ; if this was last, fix end
F466|
F466|                     ;** OK, do the processing then
F462* 02
F466| A6 EE               $2  ldx xmtATP      ; check if this is really a send
F468| BD 0002                 lda ttState,x
F46B| C9 31                   cmp #SENDRELq   ; if TRel,
F46D| F0**                   beq atXREQ      ;  skip DMAlock
F46F| 29 0F                 and #00F
F471| C9 01                   cmp #SENDREQq & 00F ; is it expected?
F473| D0DB                   bne $0      ;  if not, simply show we got to it
F475|
F475| BD 0002                 lda ttState,x   ; check again for special
F478| C9 41                   cmp #SENDNBPq   ; NBP?
F47A| F0BB                   beq atXNBP      ;  yes, do it
F47C|
F47C|                     ;** for normal usage, assume we'll need DMA anyway
F47C| A2 74                 ldx #DMAlock    ; get DMA
F47E| 20 CCE1               jsr semaP
F481| A9 C0                 lda #DMAinfo
F483| 20 24E4               jsr waitDMA
F486|
F486| A6 EE                 ldx xmtATP      ; get ptr back
F488| BD 0002                 lda ttState,x   ; REQ/RSP
F48B| C9 21                   cmp #SENDRSPq
F48D| D0**                   bne atXREQ
F48F|
F48F|                     ;** response processing.  fetch User-data, length
F48F| BD 0603             atXRSP  lda ttSqNbr,x   ; copy buffer # for later
F492| 9D 0302                 sta ttBfr,x
F495| A9 35                   lda #035        ; set FC to Host
F497| 8D 0080                 sta DMAreg
F49A| BD 0102                 lda ttRID,x     ; show which request
F49D| 8D 0080                 sta DMAreg
F4A0| BD 0302                 lda ttBfr,x     ; tell which part
F4A3|
F4A3| 8D 0080                 sta DMAreg
F4A6|
F4A6| A9 D1                   lda #DMAdata+1  ; then, wait for the data
F4A8| 20 24E4               jsr waitDMA
F4AB|
F4AB|                     ;** we now have Dlng, User data
F4AB| A6 EE                 ldx xmtATP      ; get index back
F4AD| AD 0080                 lda DMAreg
F4B0| 9D 0702                 sta ttDlng,x
F4B3| 85 E4                   sta atC
>>>>PAGE -  89  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F4B5| AD 0080                 lda DMAreg
F4B8| 9D 0802                 sta ttDlng+1,x
F4BB| 85 E5                   sta atC+1
F4BD|
F4BD| A0 04                   ldy #4      ; setup loop
F4BF| AD 0080             $1  lda DMAreg      ; copy User data
F4C2| 9D 0903                 sta ttUser,x
F4C5| E8                     inx
F4C6| 88                     dey
F4C7| D0F6                   bne $1
F4C9|
F4C9| A5 E4                   lda atC     ; check on any length
F4CB| 05 E5                   ora atC+1
F4CD| D0**                   bne atXXXY      ; share code to transfer data
F4CF| 80**                   bra atXXXZ      ;  likewise, share code for no data
F4D1|
F4D1|                     ;** note: both SENDREQq and SENDRELq take this path!!
F48D* 42
F46D* 62
F4D1| 9E 0302             atXREQ  stz ttBfr,x     ; buffer 0 (the only one) is needed
F4D4|
F4D4|                     ;** setup length info for rdDMA/XMTtask.
F4D4| BD 0702                 lda ttDlng,x
F4D7| 85 E4                   sta atC
F4D9| BD 0802                 lda ttDlng+1,x
F4DC| 85 E5                   sta atC+1
F4DE| 05 E4                   ora atC
F4E0| F0**                   beq atXXXZ      ;  and, merge
F4E2|
F4E2| A6 EE               $1  ldx xmtATP      ; reload base
F4E4| A9 33                   lda #033        ; show Request for Req data
F4E6| 8D 0080                 sta DMAreg
F4E9| BD 0102                 lda ttRID,x     ; show which request
F4EC| 8D 0080                 sta DMAreg
F4EF| BD 0302                 lda ttBfr,x     ; tell which part
F4F2| 8D 0080                 sta DMAreg
```

```
F4F5|
F4F5|                   ;** common path to read user's encapsulated data
F4CD* 26
F4F5| A9 D1             atXXXY  lda #DMAdata+1  ; then, wait for the data
F4F7| 20 1AE4               jsr waitDMA1    ;  allowing us to overlap
F4FA|
F4FA| 18                   clc         ; compute ptr for rdDMA
F4FB| A5 E0                lda atP
F4FD| 69 04
F4FF| 85 18                sta dP
F501| A5 E1                lda atP+1
F503| 85 19                sta dP+1
F505| A5 E4                lda atC     ; copy DMA stuff
F507| 85 1A                sta dC
F509| A5 E5                lda atC+1
F50B| 85 1B                sta dC+1
F50D|
F4E0* 2B
F4CF* 3C
>>>>PAGE -  90  PCCARD    FILE: PCATP.a65  PCcard - firmware for IBM-PC AppleTalk


F50D| A6 EE             atXXXZ  ldx xmtATP      ; make sure of X
F50F| 20 C8F1               jsr bldHdr      ; build header stuff
F512|
F512| A0 02                ldy #2      ; setup length field for ATPxmit
F514| A5 E4                lda atC
F516| 91 E0                sta @atP,y
F518| C8                   iny
F519| A5 E5                lda atC+1
F51B| 91 E0                sta @atP,y
F51D| 05 E4                ora atC     ; any data?
F51F| F0**                 beq $1      ;  if not, skip DMA crap
F521|
F521| A9 D1                lda #DMAdata+1  ; else, sync w/ Host
F523| 20 29E4               jsr waitDMA2
F526| 20 50E4               jsr rdDMA       ; read the data segment
F529|
F51F* 08
F529| A5 77             $1  lda DMAlock+sOwn    ; do we own DMA?
F52B| C9 58                cmp #ATPtcb     ; if not,
F52D| D0**                 bne $2      ;  skip its de-allocation
F52F| 20 3BE4               jsr freeDMA     ; else, give up DMA
F532|
F52D* 03
F532| A5 5C             $2  lda ATPBufN     ; add to xmitQ
F534| 20 80E2               jsr putXmit
F537| 4C 9DF2               jmp atpLoop     ; wait for completion
F53A|
F53A|                   ;*****************************************************************************
F53A|                   ;***    EOF, PCATP
F53A|
F53A|
F53A|
F53A|                   ;*****************************************************************************
F53A|                   ;*  The following ASCII code is inserted at this point for a "hard-
F53A|                   ;* coded" version of the copyright notice:
F53A|                   ;*****************************************************************************
F53A| 28 43 29 20 41         .ascii  "(C) Apple Computer, Inc., 1985, 1986, 1987, 1988"
F56A| 0D                     .byte   0OD
F56B| 41 6C 6C 20 72         .ascii  "All rights reserved."
F57F| 0D                     .byte   0OD
F580| 41 70 70 6C 65         .ascii  "Apple Part Number 342-0007-B14"
F59E| 0D                     .byte   0OD
F59F| 52 6F 6E 20 48         .ascii  "Ron Hochsprung, 08/10/88"
>>>>PAGE -  91  PCCARD    FILE:   PCcard - firmware for IBM-PC AppleTalk


F5A6| 68 73 70 72 75         .ascii  "Ron Hochsprung, 08/10/88" D 14
F5B7| 0D                     .byte   0OD
F5B8|
F5B8|                       .if fROM
F5B8|                       .include    pcDBG       ; debugging stuff
F5B8|                   ;*  pcDBG, debugging stuff
F5B8|                   ;*****************************************************************************
F5B8|                   ;*  pcDBG, a simple talker between PCcard and debugger.  Note that we can
F5B8|                   ;* get here by using NMI.  We attempt to save the state of the 65C02 in case
F5B8|                   ;* we wish to continue.
F5B8|                   ;*****************************************************************************
F5B8|
F5B8| FF FF FF FF FF         .org    0FE00           ; force new page boundary for table
FE00|                   DBGiTbl     ;** SCC channel-A initialization
FE00| 09 80                .byte   09, 080         ; reset channel (A)
FE02| 04 4C                .byte   04, 04C         ; 16x, 2-stop, no parity
FE04| 01 00                .byte   01, 000         ; no ints
FE06| 03 C1                .byte   03, 0C1         ; 8-bit Rx, Rx enable
FE08| 05 6A                .byte   05, 06A         ; 8-bit Tx, Tx enable, RTS
FE0A| 0B 50                .byte   0B, 050         ; Tx, Rx BRG
```

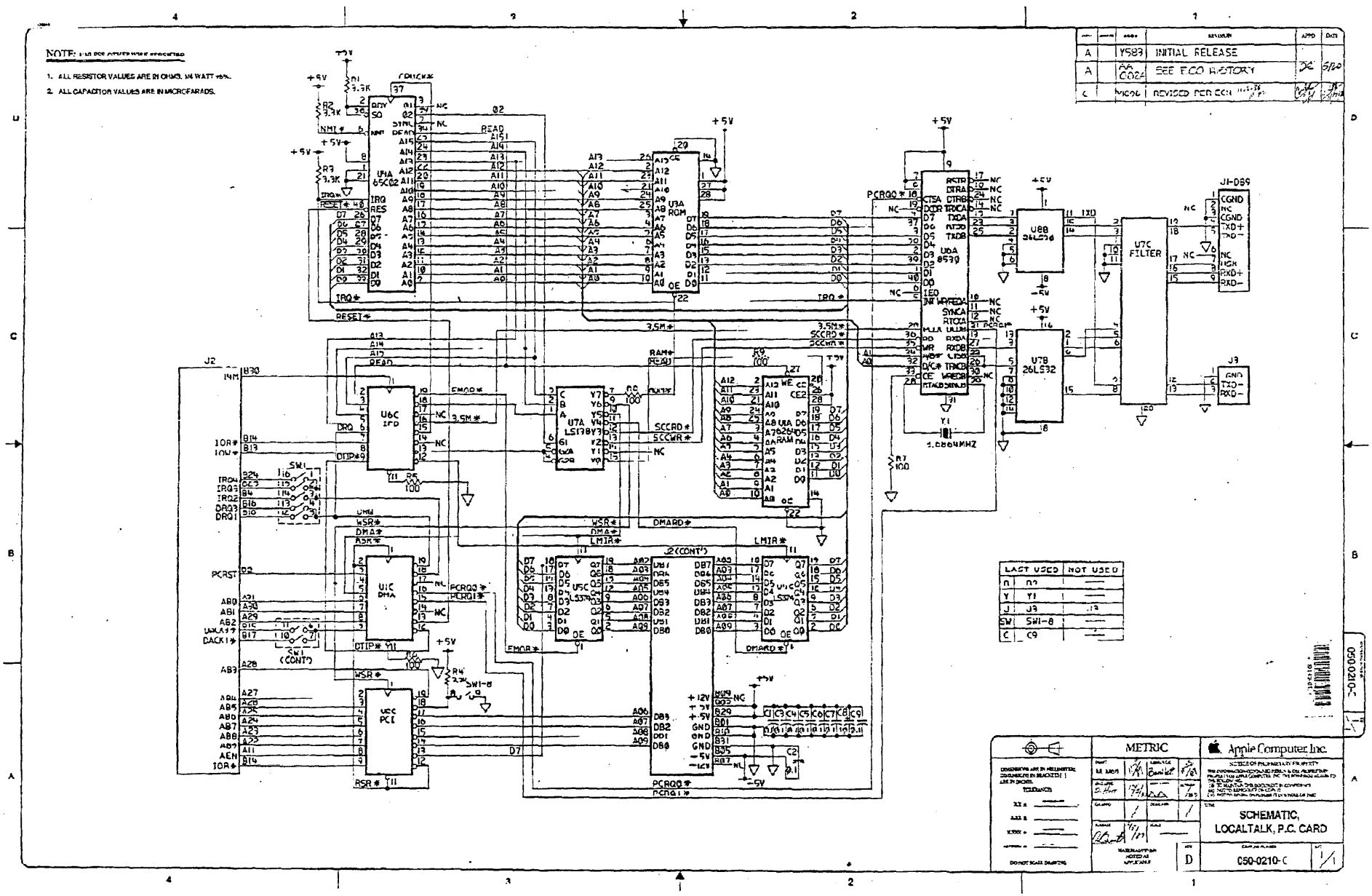
```
FE0C| 0C 0A               .byte   0C, 10.         ; 9600 baud
FE0E| 0D 00               .byte   0D, 0
FE10| 0E 03               .byte   0E, 003         ; BRG enable
FE12| 0F 00               .byte   0F, 0           ; no ext ints
FE14| 000A             DBGiTblL    .equ    $-DBGiTbl/2
FE14|
FE14|                  ;*************************************************************************
FE14|                  xGo ;* we get here when user has requested a go (or cont).  if the value of
FE14|                      ;* tAdr is -1, then it is a cont, else, we use the value in tPC
FE14| 20 ****              jsr getEOP          ; make sure it looks valid
FE17|
FE17|
FE17| A2 27                ldx #aTbl           ; reset port-A for default
FE19| A0 1A                ldy #aTblL
FE1B| 20 72E0              jsr xSCCA
FE1E|
FE1E| A9 FF                lda #0FF            ; check tAdr value
FE20| C5 F2                cmp tAdr
FE22| D0**                 bne $1
FE24| C5 F3                cmp tAdr+1
FE26| F0**                 beq xRTI            ; if FFFF, use current value
FE28|
FE22* 04
FE28| A5 F2             $1  lda tAdr            ; else, use value in tAdr
FE2A| 85 FC                sta PCreg
FE2C| A5 F3                lda tAdr+1
FE2E| 85 FD                sta PCreg+1
FE30|
FE30|                  ;*************************************************************************
FE26* 08
FE30|                  xRTI    ;* xRTI is a command which allows us to startup code after NMI (or
FE30|                      ;* break
FE30|                  ;*************************************************************************
>>>>PAGE -  92  PCCARD    FILE: PCDBG.a65  PCcard - firmware for IBM-PC AppleTalk


FE30| A6 F7                ldx SPreg           ; restore state of stack
FE32| 9A                   txs
FE33| A5 FD                lda PCreg+1         ; setup stack for RTI
FE35| 48                   pha
FE36| A5 FC                lda PCreg
FE38| 48                   pha
FE39| A5 FB                lda Preg
FE3B| 48                   pha
FE3C| A6 F8                ldx Xreg
FE3E| A4 F9                ldy Yreg
FE40| A5 FA                lda Areg
FE42| 40                   rti             ; and, pretend it never happened
FE43|
FE43|                  ;*************************************************************************
FE43|                  xNMI    ;* xNMI gets invoked by NMI interrupt.  save state and merge
FE43|                  ;*************************************************************************
FE43| 46 FF                lsr NMIflg          ; test if this is "first" time
FE45| B0**                 bcs $1          ; yes, enter delay loop
FE47| 40                   rti             ;  else, just leave
FE48|
FE45* 01
FE48| 85 FA             $1  sta Areg            ; save state
FE4A|
FE4A| 68                xBRK    pla             ; common entry for BRK (IRQ)
FE4B| 85 FB                sta Preg            ; Proc status
FE4D| 68                   pla
FE4E| 85 FC                sta PCreg
FE50| 68                   pla
FE51| 85 FD                sta PCreg+1
FE53| 84 F9                sty Yreg
FE55| 86 F8                stx Xreg            ; save other regs
FE57| BA                   tsx             ; copy SP
FE58| 86 F7                stx SPreg
FE5A|
FE5A|                  ;*************************************************************************
FE5A|                  ;*  basic debug interaction here
FE5A|                  ;*************************************************************************
FE5A| D8                xTalk   cld             ; make sure of state
FE5B| A0 0A                ldy #DBGiTblL           ; loop counter
FE5D| A2 00                ldx #0          ; index
FE5F| AD 0240              lda DBGctrl         ; make sure we're in step
FE62|
FE62| BD 00FE          $1  lda DBGiTbl,x       ; pickup reg#
FE65| 8D 0240              sta DBGctrl
FE68| E8                   inx
FE69| BD 00FE              lda DBGiTbl,x       ; value
FE6C| 8D 0240              sta DBGctrl
FE6F| E8                   inx
FE70| 88                   dey
FE71| D0EF                 bne $1
FE73|
FE73| A2 7F            reTalk  ldx #STKbase-1 % 0100   ; re-load SP
```

```
FE75| 9A                    txs
FE76| A0 00                 ldy #0          ; set Y for first time thru
FE78|
>>>>PAGE -  93  PCCARD    FILE: PCDBG.a65  PCcard - firmware for IBM-PC AppleTalk


FE78| 84 F1              $0  sty tFunc           ; reset flags, checksum
FE7A| 84 F0                  sty tCKS
FE7C| 20 ****                jsr getB           ; get a byte
FE7F| C9 02                  cmp #aSTX          ;  correct beginning?
FE81| D0F5                   bne $0          ; nope, keep looking
FE83|
FE83|                    ;** looks like beginning of a packet, get 1st 3 bytes
FE83| A9 01                  lda #001           ; reset NMI flag
FE85| 85 FF                  sta NMIflg
FE87| 20 ****                jsr getB
FE8A| D0**                   bne $1          ; if non-valid FC
FE8C| 4C ****                jmp xNAK           ;  send a NAK and re-examine
FE8F|
FE8A* 03
FE8F| 85 F1              $1  sta tFunc           ; else, save the function code
FE91|
FE91| 20 ****                jsr getB           ; get tAdr
FE94| 85 F3                  sta tAdr+1         ;  (in reverse order)
FE96| 20 ****                jsr getB
FE99| 85 F2                  sta tAdr
FE9B|
FE9B|                    ;** analyze the function
FE9B| A6 F1                  ldx tFunc          ; get it back
FE9D| CA                    dex             ; start decoding
FE9E| F0**                  beq xLoad          ; 1 -> load image
FEA0| CA                    dex
FEA1| F0**                  beq xDump          ; 2 -> get bytes
FEA3| CA                    dex
FEA4| D0**                  bne $3
FEA6| 4C 14FE               jmp xGo         ; 3 -> go
FEA4* 03
FEA9| 4C ****            $3  jmp xNAK           ; if none, error
FEAC|
FEAC|                    ;*****************************************************************************
FE9E* 0C
FEAC|                    xLoad   ;* load a (large) group of bytes.
FEAC|                    ;*****************************************************************************
FEAC| 20 ****                jsr getB           ; get word count
FEAF| 85 F5                  sta tCnt+1         ;  (reverse order)
FEB1| 20 ****                jsr getB
FEB4| 85 F4                  sta tCnt
FEB6| D0**                   bne $1          ; if mod 256
FEB8| C6 F5                   dec tCnt+1         ;  special case it
FEBA|
FEB6* 02
FEBA| 20 ****            $1  jsr getB           ; get the next byte
FEBD| 91 F2                  sta @tAdr,Y        ; do it
FEBF| 20 ****                jsr incAdr
FEC2| 90F6                   bcc $1
FEC4|
FEC4| 20 ****                jsr getEOP         ; check all at end
FEC7|
FEC7|                    ;** acknowledge a correct outward packet
FEC7| A9 06              xACK    lda #aACK           ; send ACK for good feelings
FEC9| 20 ****                jsr putB
>>>>PAGE -  94  PCCARD    FILE: PCDBG.a65  PCcard - firmware for IBM-PC AppleTalk


FECC| 4C 73FE                jmp reTalk         ;  and loop
FECF|
FECF|                    ;*****************************************************************************
FEA1* 2C
FECF|                    xDump   ;* get a bunch of bytes back.
FECF|                    ;*****************************************************************************
FECF|
FECF| 20 ****                jsr getB           ; count
FED2| 85 F5                  sta tCnt+1
FED4| 20 ****                jsr getB
FED7| 85 F4                  sta tCnt
FED9| D0**                   bne $1          ; if mod 256
FEDB| C6 F5                   dec tCnt+1         ;  special case it
FEDD|
FED9* 02
FEDD| 20 ****            $1  jsr getEOP         ; end-of-packet
FEE0| A9 00                  lda #0          ; start checksum
FEE2| 85 F0                  sta tCKS
FEE4| A9 02                  lda #aSTX          ; start reply
FEE6| 20 ****                jsr putB
FEE9|
FEE9| B1 F2              $2  lda @tAdr,Y        ; send the bytes requested
FEEB| 20 ****                jsr putB
FEEE| 20 ****                jsr incAdr
```

```
FEF1| 90F6                bcc $2          ; loop till done
FEF3|
FEF3| 20 ****             jsr putEOP          ; end the packet
FEF6| 4C 73FE             jmp reTalk          ;  and, we're done
FEF9|
FEF9|                   ;** send NAK to let clown know somethings wrong
FEAA* F9FE
FE8D* F9FE
FEF9| A9 15             xNAK    lda #aNAK
FEFB| 20 ****             jsr putB
FEFE| 4C 73FE             jmp reTalk          ;  and loop
FF01|
FF01|                   ;** getEOP, get end-of-pkt, w/ checksum test
FEDE* 01FF
FEC5* 01FF
FE15* 01FF
FF01| 20 ****           getEOP  jsr getB            ; get presumed ETX
FF04| C9 03                cmp #aETX
FF06| D0F1                bne xNAK            ; if not correct, flag it
FF08| 20 ****             jsr getB            ; this sb CKS
FF0B| A5 F0               lda tCKS            ; is it 0 at this pt?
FF0D| D0EA                bne xNAK            ;  nope, error
FF0F| 60                  rts             ; else, simply return
FF10|
FF10|                   ;** putEOP, send a proper end-of-packet
FEF4* 10FF
FF10| A9 03             putEOP  lda #aETX           ; send ETX
FF12| 20 ****             jsr putB
FF15| 06 F0               asl tCKS            ; then, checksum
FF17| 90**                bcc $1
FF19| E6 F0               inc tCKS
FF17* 02
>>>>PAGE -  95  PCCARD    FILE: PCDBG.a65  PCcard - firmware for IBM-PC AppleTalk


FF1B| A9 00             $1  lda #0
FF1D| 38                   sec
FF1E| E5 F0               sbc tCKS            ; the negative
FF20| 20 ****             jsr putB
FF23| 60                  rts
FF24|
FF24|                   ;** getB, fetch a byte
FF09* 24FF
FF02* 24FF
FED5* 24FF
FED0* 24FF
FEBB* 24FF
FEB2* 24FF
FEAD* 24FF
FE97* 24FF
FE92* 24FF
FE88* 24FF
FE7D* 24FF
FF24| A9 01             getB    lda #RCA            ; setup test loop
FF26| A2 00                ldx #0          ; timout counters
FF28| A0 00                ldy #0
FF2A| 2C 0240           $1  bit DBGctrl         ; have a byte?
FF2D|
FF2D| D0**                bne $2          ;  yes, process it
FF2F| 88                  dey
FF30| D0F8                bne $1
FF32| CA                  dex
FF33| D0F5                bne $1
FF35|                   ;* if we timeout, reset the world
FF35| A5 F1               lda tFunc           ; are we in packet?
FF37| D0C0                bne xNAK            ;  yes, so send NAK first
FF39| 4C 73FE             jmp reTalk          ; else, just do again
FF3C|
FF2D* 0D
FF3C| AD 0340           $2  lda DBGdata         ; get the byte
FF3F| 48                   pha             ; save it
FF40| 06 F0               asl tCKS            ; accumulate Checksum
FF42| 65 F0               adc tCKS
FF44| 85 F0               sta tCKS
FF46| A0 00                ldy #0          ; leave w/ Y zero for stores
FF48| 68                  pla             ; note: N/Z are set by recv'd byte
FF49| 60                  rts
FF4A|
FF4A|                   ;** putB, stash a byte
FF21* 4AFF
FF13* 4AFF
FEFC* 4AFF
FEEC* 4AFF
FEE7* 4AFF
FECA* 4AFF
FF4A| 48                 putB    pha             ; save the byte
FF4B| A9 04                lda #TBE            ; set bit
FF4D| 2C 0240           $1  bit DBGctrl         ; ready for next?
```

```
FF50| F0FB                beq $1
>>>>PAGE -  96  PCCARD    FILE: PCDBG.a65  PCcard - firmware for IBM-PC AppleTalk


FF52| 68                  pla
FF53| 8D 0340              sta DBGdata
FF56| 06 F0                asl tCKS            ; update checksum
FF58| 65 F0                adc tCKS
FF5A| 85 F0                sta tCKS
FF5C| A0 00                ldy #0          ; make sure of Y for indirect stores
FF5E| 60                  rts
FF5F|
FF5F|                   ;** incAdr, perform updates of tAdr, tCnt;  leave C set when done
FEEF* 5FFF
FEC0* 5FFF
FF5F| E6 F2              incAdr  inc tAdr            ; update address
FF61| D0**                bne $1
FF63| E6 F3                inc tAdr+1
FF61* 02
FF65| C6 F4              $1  dec tCnt            ; dec counter
FF67| D0**                bne $2
FF69| C6 F5                dec tCnt+1
FF6B| 10**                bpl $2
FF6D| 38                  sec             ; show done
FF6E| 60                  rts
FF6F|
FF6B* 02
FF67* 06
FF6F| 18                $2  clc             ; signal more to go
FF70| 60                  rts             ;  and, leave
FF71|
FF71|                   ;***    eof, pcDBG
FF71|
FF71|
FF71| FF FF FF FF FF        .org    0FFFA       ; vectors for code
FFFA| 43FE                .word   xNMI
FFFC| 94E0                .word   xRESET      ; hardware RESET vector
FFFE| ACE2                .word   SCCint
0000|                     .endc
0000|
0000|                     .end
>>>>PAGE -  97  PCCARD    FILE:   SYMBOLTABLE DUMP


AB - Absolute     LB - Label     UD - Undefined     MC - Macro
RF - Ref          DF - Def       PR - Proc          FC - Func
PB - Public       PV - Private   CS - Consts


AACK     AB 0006|  ABTRQRS  LB F044|  ADDDELE  LB E7CD|  ADDDELW  LB E7C8|  ADDDELX  LB E7C0
AETX     AB 0003|  AEXT     LB E2CA|  ANAK     AB 0015|  AREG     AB 00FA|  ASTX     AB 0002
ATBE     LB E295|  ATBITS   LB E05C|  ATBL     AB 0027|  ATBLL    AB 001A|  ATC      AB 00E4
ATMAPS   LB E053|  ATP      AB 00E0|  ATPH     AB 00E2|  ATPLOOP  LB F29D|  ATPSCAN  LB F373
ATPSCN1  LB F381|  ATPSCN2  LB F3AE|  ATPTASK  LB F283|  ATPTIME  LB F2A0|  ATPXMIT  LB F440
ATSNBPZ  LB F33F|  ATSREL   LB F35C|  ATSRELZ  LB F34D|  ATSREQ   LB F3BF|  ATSREQZ  LB F349
ATSRSP   LB F3DE|  ATSRSPX  LB F3D9|  ATSRSPZ  LB F349|  ATSXXX   LB F3C2|  ATSZZZZ  LB F34F
ATTXQ    AB 00EC|  ATTXQHD  AB 00EC|  ATTXQTL  AB 00ED|  ATXNBP   LB F437|  ATXREQ   LB F4D1
ATXRSP   LB F48F|  ATXXXY   LB F4F5|  ATXXXZ   LB F50D|  BADCT    AB 00C2|  BBIT     AB 0010
BEXT     LB E2DA|  BITCOUNT LB EA49|  BITS     LB E04B|  BLDDDP   LB F23A|  BLDDDPX  LB F1E8
BLDHDR   LB F1C8|  BLDHDR1  LB F25D|  BLDTREQ  LB F134|  BLDTRSP  LB F11C|  BLDTT    LB F0D1
BLDTT1   LB F0EF|  BTBL     AB 0000|  BTBLL    AB 0027|  CBIT     AB 0001|  CHKDDP   LB EB49
CHKLAP   LB E7E1|  CHKSKT   LB EB78|  CKDATA   LB ECE7|  CKDATA0  LB ED0D|  CKHDR    LB ECB6
CKIDLE   LB EA32|  CKIDLE1  LB EA34|  CMDDONE  LB E345|  CMDSTAT  LB E34C|  CMDTASK  LB E35E
CRCCT    AB 00C0|  CTBL     LB E2EA|  CTBLL    AB 0039|  CTSCT    AB 00B8|  CVTBUFN  LB E244
DBGITBL  LB FE00|  DBGITBLL AB 000A|  DBIT     AB 0008|  DISPATCH LB E212|  DOCKS    LB EC97
DUPCT    AB 00C6|  FNDTT    LB EFEB|  FREEDMA  LB E43B|  GETB     LB FF24|  GETEOP   LB FF01
GETFREE  LB E233|  GOCMD    LB E3B1|  IBIT     AB 0004|  INCADR   LB FF5F|  ITBL     LB E00A
LAPTBL   AB 0160|  LNGCT    AB 00C4|  LSTATP   AB 00E9|  LSTTTP   AB 00E8|  NBIT     AB 0080
NBRTRYS  AB 00CE|  NBTMOUT  AB 00CD|  NBTTP    AB 00CC|  NMIFLG   AB 00FF|  NRXREQ   LB EF47
OPNCLSE  LB EB44|  OPNCLSX  LB EB2F|  OVRCT    AB 00BE|  PCCARD   PR ----|  PCREG    AB 00FC
POSTSTS  LB E323|  PREG     AB 00FB|  PUTB     LB FF4A|  PUTEOP   LB FF10|  PUTFREE  LB E273
PUTXMIT  LB E280|  RANDOM   LB EA58|  RCVFREE  LB E4CA|  RCVTASK  LB E4CD|  RCVWAIT  LB E4E3
RDDMA    LB E450|  RETALK   LB FE73|  RQBASE   AB 0309|  RQPAGE   AB 0300|  RTSCT    AB 00B6
RTXMIT   LB EE0D|  RXACK    LB E622|  RXATP    LB EEC5|  RXBAD    LB E62C|  RXCRC    LB E5E8
RXCTS    LB E622|  RXDATA   LB E6B4|  RXDDP1   LB EAD5|  RXENB    AB 0022|  RXENBL   AB 0003
RXENQ    LB E680|  RXEOF    LB E5F5|  RXFLUSH  LB E653|  RXINTR   LB E5A3|  RXLAP    LB E60A
RXLNG    LB E5E4|  RXOVR    LB E5E0|  RXREL    LB EE9B|  RXREQ    LB EEE4|  RXRESET  LB E62E
RXRSP    LB EF58|  RXRST    AB 0020|  RXRSTL   AB 0005|  RXRTI    LB E648|  RXRTS    LB E684
RXSKP    LB E5ED|  RXSKP1   LB E5EC|  RXSTS    LB EF12|  RXUND    LB E5F1|  RXXXX    LB E68A
SCCINT   LB E2AC|  SCCINTX  LB E2AD|  SEMAP    LB E1CC|  SEMAV    LB E198|  SEMAX    LB E1CA
SENDNBP  AB 0040|  SENDNBPN AB 0046|  SENDNBPX AB 0042|  SENDNBPZ AB 0045|  SENDREL  AB 0030
SENDRELR AB 0033|  SENDRELX AB 0034|  SENDRELZ AB 0032|  SENDREQ  AB 0010|  SENDREQX AB 0012
SENDREQZ AB 0015|  SENDRSP  AB 0020|  SENDRSPN AB 0028|  SENDRSPR AB 0026|  SENDRSPX AB 0022
SENDRSPZ AB 0025|  SETDPDC  LB E14A|  SETRTMP  LB EDB1|  SETRXP   LB E25B|  SETSTAT  LB E080
SIGNAL   LB E174|  SKPCT    AB 00BA|  SKTTBL   AB 0140|  SNDNBPE  LB EB90|  SPREG    AB 00F7
STCKS    LB ED27|  STKBASE  AB 0180|  STKSIZE  AB 0020|  STSTBL   AB 0100|  TADR     AB 00F2
TASKP    LB E000|  TCKS     AB 00F0|  TCNT     AB 00F4|  TFUNC    AB 00F1|  TSTKP    AB 00F6
TTBASE   AB 0209|  TTBFR    AB 0203|  TTBTMAP  AB 0306|  TTBUFN   AB 0204|  TTCCI    AB 0305
```

```
TTDLNG   AB 0207|  TTDNET   AB 0300|  TTDNODE  AB 0302|  TTDSKT   AB 0303|  TTFLGS   AB 0204
TTLINK   AB 0202|  TTNBR    AB 0013|  TTNEXT   AB 030D|  TTPAGE   AB 0200|  TTRID    AB 0201
TTRSMAP  AB 0205|  TTRTRYS  AB 0206|  TTSIZE   AB 000D|  TTSIZET  AB 00F7|  TTSQNBR  AB 0306
TTSSKT   AB 0304|  TTSTATE  AB 0200|  TTTID    AB 0307|  TTTMOUT  AB 0205|  TTTMR    AB 0203
TTTXMAP  AB 0206|  TTUSER   AB 0309|  TXCOLSN  LB E951|  TXDDP    LB EC2E|  TXDDP1   LB EC54
TXDDP2   LB ED30|  TXDDPX   LB EC1C|  TXDEFER  LB E8E6|  TXFFCS   LB E9B3|  TXFHDR   LB E9EF
TXFRAME  LB E9B0|  TXIDLWT  LB E8A1|  TXN1     LB ED5E|  TXNBP    LB ED44|  TXPACKET LB E857
TXPKT0   LB E897|  TXPKT1   LB E8BE|  TXPKT2   LB E8F6|  TXPKT3   LB E904|  TXPKT4   LB E927
TXPKT5   LB E969|  TXPKTAE  LB E59C|  TXPKTX   LB E9AD|  UNDCT    AB 00BC|  VBIT     AB 0040
WAIT     LB E164|  WAITDMA  LB E424|  WAITDMA2 LB E429|  WRDMA    LB E48D|  XABTNBP  LB F03A
XABTREQ  LB F03E|  XABTRSP  LB F042|  XACK     LB FEC7|  XADDLAP  LB E7B3|  XADDRSP  LB F17C
>>>>PAGE -  98  PCCARD    FILE:   SYMBOLTABLE DUMP


XBRIDGE  LB E401|  XBRK     LB FE4A|  XC       AB 00D2|  XCHIST   AB 00D4|  XCLSSKT  LB EB37
XCTRYS   AB 00D5|  XDELAY   AB 00DD|  XDELLAP  LB E7D2|  XDHIST   AB 00D6|  XDTRYS   AB 00D7
XDUMP    LB FECF|  XFBCAST  AB 00DB|  XGMASK   AB 00D8|  XGO      LB FE14|  XINITLAP LB E735
XLMASK   AB 00D9|  XLOAD    LB FEAC|  XMTATP   AB 00EE|  XMTTASK  LB E528|  XNAK     LB FEF9
XNMI     LB FE43|  XOPNSKT  LB EB24|  XP       AB 00D0|  XPC2US   LB E130|  XPCERR   LB E3AC
XRDSFLG  LB E3FC|  XREG     AB 00F8|  XRESET   LB E094|  XRTI     LB FE30|  XRXATP   LB EE70
XRXATPX  LB EE3E|  XRXDDP   LB EAA5|  XRXDDP1  LB EACB|  XRXDDPX  LB EA84|  XRXLAP   LB E6E7
XRXPKT1  LB E72A|  XRXPKT2  LB E71F|  XRXRTMP  LB ED91|  XSCCA    LB E072|  XSCCB    LB E064
XSEED    AB 00DE|  XSNDNBP  LB EB8C|  XSNDREQ  LB F0CF|  XSNDRSP  LB F0A5|  XSTATUS  LB E3BC
XTALK    LB FE5A|  XTXDDP   LB EB95|  XTXLAP   LB E7F1|  XTXPKTA  LB E585|  XTXPKTD  LB E82B
XTXPKTX  LB E812|  XTXPKTZ  LB E823|  XUS2PC   LB E13D|  XWRSFLG  LB E3F7|  YIELD    LB E1F7
YIELD1   LB E20B|  YREG     AB 00F9|  ZBIT     AB 0002|
>>>>PAGE -  99  PCCARD    FILE:   PCcard - firmware for IBM-PC AppleTalk


Current minimum space is      450 words.
E161* 24E4
E136* 50E4
E143* 8DE4
E006* CCE4
E004* 27E5
E396* 35E7
E2FA* B2E7
E2FD* D1E7
E300* F0E7
E571* 57E8
E770* 58EA
E526* 84EA
E51D* A5EA
E303* 23EB
E306* 36EB
E30F* 8BEB
E30C* 94EB
E309* 94EB
E7B1* B1ED
EA9F* 3EEE
EAC7* 70EE
E312* 39F0
E31E* 3DF0
E321* 41F0
E318* A4F0
E315* CEF0
E31B* 7BF1
E008* 82F2

Assembly complete:      4947 lines
     0   Errors flagged on this Assembly
```


NOTE:
1. ALL RESISTOR VALUES ARE IN OHMS, 1/4 WATT ±5%.
2. ALL CAPACITOR VALUES ARE IN MICROFARADS.
A | Y583 | INITIAL RELEASE
A | AA C024 | SEE ECO HISTORY
C | M696 | REVISED PER ECN
U6A 65C02
U3A ROM
U7A LS138
U6C IFD
U1C DMA
U5C LS374
U7A VIA
U8A 8530
U8B 26LS32
U7B 26LS32
U7C FILTER
Y1 3.6864MHZ
J1-DB9
J3
J2
SW1
SW1-8
LAST USED | NOT USED
METRIC
Apple Computer Inc.
SCHEMATIC, LOCALTALK, P.C. CARD
D
050-0210-C