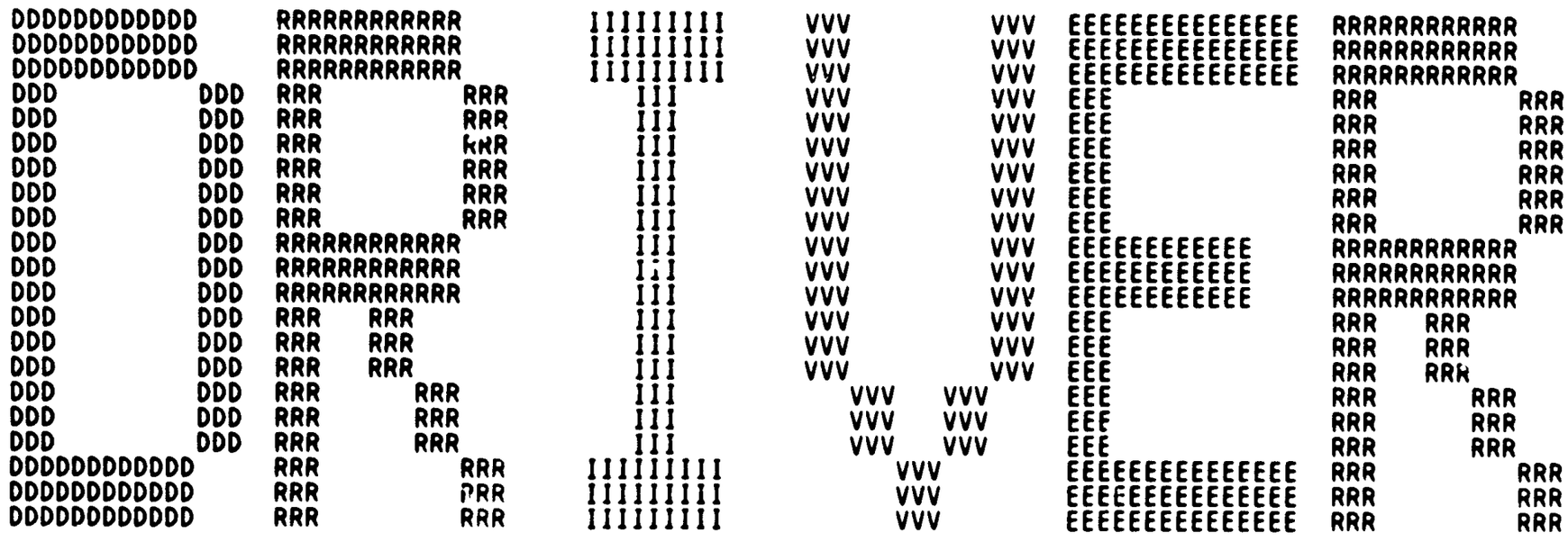
DDDDDDDDDDDD RRRRRRRRRRRR IIIIIIIII VVV VVV EEEEEEEEEEEEEEE RRRRRRRRRRRR
DDDDDDDDDDDD RRRRRRRRRRRR IIIIIIIII VVV VVV EEEEEEEEEEEEEEE RRRRRRRRRRRR
DDDDDDDDDDDD RRRRRRRRRRRR IIIIIIIII VVV VVV EEEEEEEEEEEEEEE RRRRRRRRRRRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRRRRRRRRRRR III VVV VVV EEEEEEEEEEEE RRRRRRRRRRRR
DDD DDD RRRRRRRRRRRR III VVV VVV EEEEEEEEEEEE RRRRRRRRRRRR
DDD DDD RRRRRRRRRRRR III VVV VVV EEEEEEEEEEEE RRRRRRRRRRRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDD DDD RRR RRR III VVV VVV EEE RRR RRR
DDDDDDDDDDDD RRR RRR IIIIIIIII VVV EEEEEEEEEEEEEEE RRR RRR
DDDDDDDDDDDD RRR RRR IIIIIIIII VVV EEEEEEEEEEEEEEE RRR RRR
DDDDDDDDDDDD RRR RRR IIIIIIIII VVV EEEEEEEEEEEEEEE RRR RRR

```
XX      XX    QQQQQQ    DDDDDDDD    RRRRRRRR      IIIIII    VV      VV  EEEEEEEEEE  RRRRRRRR
XX      XX    QQQQQQ    DDDDDDDD    RRRRRRRR      IIIIII    VV      VV  EEEEEEEEEE  RRRRRRRR
XX      XX  QQ      QQ  DD      DD  RR      RR      II      VV      VV  EE          RR      RR
XX      XX  QQ      QQ  DD      DD  RR      RR      II      VV      VV  EE          RR      RR
 XX  XX     QQ      QQ  DD      DD  RR      RR      II      VV      VV  EE          RR      RR
 XX  XX     QQ      QQ  DD      DD  RR      RR      II      VV      VV  EE          RR      RR
   XX       QQ      QQ  DD      DD  RRRRRRRR        II      VV      VV  EEEEEEEE    RRRRRRRR
   XX       QQ      QQ  DD      DD  RRRRRRRR        II      VV      VV  EEEEEEEE    RRRRRRRR
 XX  XX     QQ  QQ  QQ  DD      DD  RR  RR          II      VV      VV  EE          RR  RR
 XX  XX     QQ  QQ  QQ  DD      DD  RR  RR          II      VV      VV  EE          RR  RR
XX      XX   QQ    QQ   DD      DD  RR    RR        II        VV  VV    EE          RR    RR        ....
XX      XX   QQ    QQ   DD      DD  RR    RR        II        VV  VV    EE          RR    RR        ....
XX      XX     QQQQ QQ  DDDDDDDD    RR      RR    IIIIII        VV      EEEEEEEEEE  RR      RR      ....
XX      XX     QQQQ QQ  DDDDDDDD    RR      RR    IIIIII        VV      EEEEEEEEEE  RR      RR      ....

LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

Table of contents

```
(67)    7111        ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
(68)    7182        MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
(69)    7212        ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
(70)    7248        VALIDATE_P2 - VALIDATE P2 BUFFER PARAMETERS
(71)    7417        CHANGE_PARAM - UPDATE UCB/CDB BASED ON P2 BUFFER PARAMETERS
(72)    7546        RETURN_P2,  Return UCB/CDB buffer parameters
(73)    7654        VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST
(74)    7711        VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
(75)    7773        SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
(76)    7927        SET_PHYAD - SET THE PHYSICAL ADDRESS
(76)    7928        SET_DESAD - SET THE DESTINATION ADDRESS
(77)    8009        RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
(78)    8052        MATCH_MULTI - CHECK MULTICAST ADDRESS
(79)    8097        MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
(80)    8139        POKE_USER - DELIVER ATTENTION ASTS
(81)    8181        MATCH_PROTYP - Match protocol type
(81)    8182        MATCH_PROMTYP - Find the promiscuous user
```

```
                     0000     1          .TITLE  XQDRIVER - VAX/VMS QNA driver
;RNG0001             0000    .1          .IDENT  'V04-001'
;RNG0001             0000    .2          .NLIST  CND
-1                   0000     3 ;
                     0000     4 ;*****************************************************************************
                     0000     5 ;*                                                                           *
                     0000     6 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
                     0000     7 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                    *
                     0000     8 ;*  ALL RIGHTS RESERVED.                                                     *
                     0000     9 ;*                                                                           *
                     0000    10 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
                     0000    11 ;*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE            *
                     0000    12 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER       *
                     0000    13 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
                     0000    14 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY        *
                     0000    15 ;*  TRANSFERRED.                                                             *
                     0000    16 ;*                                                                           *
                     0000    17 ;*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE      *
                     0000    18 ;*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT         *
                     0000    19 ;*  CORPORATION.                                                             *
                     0000    20 ;*                                                                           *
                     0000    21 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS      *
                     0000    22 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
                     0000    23 ;*                                                                           *
                     0000    24 ;*                                                                           *
                     0000    25 ;*****************************************************************************
                     0000    26 ;++
                     0000    27 ; FACILITY:
                     0000    28 ;
                     0000    29 ;       VAX/VMS DEQNA QUEUE I/O DRIVER
                     0000    30 ;
                     0000    31 ; ABSTRACT:
                     0000    32 ;
                     0000    33 ;       This module contains the DEQNA driver FDT routines,
                     0000    34 ;       interrupt dispatcher, interrupt service and fork routines.
                     0000    35 ;
                     0000    36 ; AUTHOR:
                     0000    37 ;
                     0000    38 ;       Rod Gamache     26-Jul-1983
                     0000    39 ;
                     0000    40 ;
                     0000    41 ; MODIFICATION HISTORY:
                     0000    42 ;
;RNG0001             0000    .1 ;       V04-001 RNG0001         Rod Gamache             1-Oct-1984
;RNG0001             0000    .2 ;               Fix additional buffer quota test.
;RNG0001             0000    .3 ;               Add support for MicroVAX II.
;RNG0001             0000    .4 ;               Add some performance enhancements similar to those
;RNG0001             0000    .5 ;               added to the DEUNA device driver.
;RNG0001             0000    .6 ;               Fix loopback mode.
;RNG0001             0000    .7 ;               Get vector address from IDB.
;RNG0001             0000    .8 ;               Fix receive length word set up.
;RNG0001             0000    .9 ;               Fix timeout errors.
;RNG0001             0000   .10 ;
-120                 0000   163 ;--
                     0000   164
                     0000   165 ;
                     0000   166 ; EXTERNAL SYMBOLS
```

```
                     0000    167 ;
                     0000    168         $ABDDEF                         ; Define ABDs
                     0000    169         $ACBDEF                         ; Define AST control block
                     0000    170         $CANDEF                         ; Define CANCEL reason codes
                     0000    171         $CCBDEF                         ; Define CCB offsets
                     0000    172         $CRBDEF                         ; Define CRB
                     0000    173         $CXBDEF                         ; Define CXB
                     0000    174         $DCDEF                          ; Define device classes and types
                     0000    175         $DDBDEF                         ; Define DDB
                     0000    176         $DEVDEF                         ; Define device characteristics
                     0000    177         $DPTDEF                         ; Define DPT
                     0000    178         $DYNDEF                         ; Define dynamic data structures
                     0000    179         $FFIDEF                         ; Fast Interface definitions
                     0000    180         $FKBDEF                         ; Fork block definitions
                     0000    181         $IDBDEF                         ; Define IDB
                     0000    182         $IODEF                          ; Define I/O function codes
                     0000    183         $IPLDEF                         ; Define IPLs
                     0000    184         $IRPDEF                         ; Define IRP
                     0000    185         $JIBDEF                         ; Define JIB
                     0000    186         $MSGDEF                         ; Define SYSTEM MESSAGES
                     0000    187         $NMADEF                         ; Define Network Management Codes
                     0000    188         $ORBDEF                         ; OBJECT'S RIGHTS BLOCK OFFSETS
                     0000    189         $PCBDEF                         ; Define PCB
                     0000    190         $PRDEF                          ; Processor register definitions
                     0000    191         $PRVDEF                         ; Privilege bit definitons
                     0000    192         $PTEDEF                         ; Define system PTEs
                     0000    193         $SSDEF                          ; Define System Status Codes
                     0000    194         $TQEDEF                         ; Define TQE offsets
                     0000    195         $UBADEF                         ; Define UBA symbols
                     0000    196         $UCBDEF                         ; Define UCB
                     0000    197         $VADEF                          ; Define Virtual Address bits
                     0000    198         $VECDEF                         ; Define CRB VECTOR
                     0000    199         $XMDEF                          ; Define DECnet datalink characteristics
                     0000    200 ;
                     00C0    201 ; Local symbol definitions
                     0000    202 ;
                     0000    203
                     0000    204 ;
                     0000    205 ; Define the following symbol to enable use of point-to-point mode
                     0000    206 ;
;RNG0001              0000      .1 ;       POINT = 1                       ; Enable use of point-to-point mode
;RNG0001              0000      .2 ;
;RNG0001              0000      .3 ; Define the following symbol to change the RING INVALIDATION test to
;RNG0001              0000      .4 ; perform POLLing of the CSR's.
;RNG00C1              0000      .5 ;
;RNG0001              0000      .6 ;       POLL = 1                        ; Enable POLL scanning of CSR's
-1                   0000    208
                     0000    209 ;
                     0000    210 ; Argument list offsets for QIO
                     0000    211 ;
00000000             0000    212 P1      = 0                             ; Parameter 1
00000004             0000    213 P2      = 4                             ; Parameter 2
00000008             0000    214 P3      = 8                             ; Parameter 3
0000000C             0000    215 P4      = 12                            ; Parameter 4
00000010             0000    216 P5      = 16                            ; Destination/Source address
                     0000    217
                     0000    218 .
```

```
                     0000   219 ; Constants
                     0000   220 ;
                     0000   221 $EQU    BRDCST1         <^XFFFFFFFF>    ; Broadcast address
                     0000   222 $EQU    BRDCST2         <^XFFFF>        ;       ...
                     0000   223 $EQU    MAX_C_MLT       12              ; Maximum number of multicast addresses
                     0000   224 $EQU    MAX_C_XMT       4               ; Maximum number of entries in XMT ring
                     0000   225 $EQU    MAX_C_RCV       8               ; Maximum number of entries in RCV ring
                     0000   226 $EQU    MAX_C_XMTUV1    1               ; Maximum number of XMTs on micro-VAX I
                     0000   227 $EQU    MAX_C_RCVUV1    5               ; Maximum number of RCVs on micro-VAX I
                     0000   228 $EQU    MAX_PKT_SIZE    1500            ; Size of maximum Ethernet user data
                     0000   229 $EQU    XQ_C_HEADER     14              ; Size of Ethernet header
                     0000   230 $EQU    XQ_C_CRC        4               ; Size of Ethernet CRC
                     0000   231 $EQU    XQ_C_CNTSIZ     2               ; Size of packet count field
                     0000   232 $EQU    MAX_BUFSIZ_UV1  MAX_PKT_SIZE+XQ_C_HEADER
                     0000   233 $EQU    UV1_BUFFER_AREA <<MAX_C_XMTUV1+MAX_C_RCVUV1>*MAX_BUFSIZ_UV1>
                     0000   234 $EQU    MAX_C_CHAIR     1               ; Maximum number of extra segments in
                     0000   235                                         ;  a receive buffer chain
                     0000   236
                     0000   237 $EQU    NI_CTR_PROTYP   <^X0260>        ; Ethernet read counters protocol 60-02
                     0000   238 $EQU    NI_CTR_READ     9               ; Read counters request function
                     0000   239 $EQU    NI_CTR_REPLY    11              ; Read counters reply function
                     0000   240
;RNG0001              0000    .1 $EQU    INIT_C_QUOTA    6*MAX_PKT_SIZE  ; Allow for 6 of the largest buffers
-1                   0000   242 $EQU    INIT_C_AQUOTA   2               ; Allow for 2 additional buffers
                     0000   243 $EQU    INIT_C_BUFSIZE  128             ; Size of init (setup mode) buffer
                     0000   244 $EQU    DSC$A_POINTER   4               ; Pointer to data in buffer descriptor
                     0000   245 $EQU    MIN_PKT_SIZE    46              ; Size of user data in a runt packet
                     0000   246 $EQU    TQE_C_DELTA     2               ; 2 second timer interval
                     0000   247 $EQU    TQE_DELTA       TQE_C_DELTA*10000*1000 ; Delta interval (in 100 nsec)
                     0000   248 $EQU    RESTART_DELTA   3*10000*1000    ; RESTART interval - 3 Seconds
;RNG0001              0000    .1 $EQU    XMT_C_TIM       6               ; XMITS must take less than 6 seconds
;RNG0001              0000    .2 $EQU    XMT_TIM         <<XMT_C_TIM*2>/2>
-4                   0000   253 $EQU    XQ_C_ADDRCV     64              ; Size to add to received packets
                     0000   254 $EQU    XQ_C_STPRO      <^X0660>        ;% PT-TO-PT Starting protocol type 60-06
                     0000   255 $EQU    IPL$_XQ_FIPL    8               ; Fork IPL
                     0000   256 $EQU    IPL$_XQ_DIPL    21              ; Device IPL
                     0000   257
                     0000   258 ;
                     0000   259 ; Local macros
                     0000   260 ;
                     0000   261 .MACRO  SETBIT  VAL,FLAG
                     0000   262         .NTYPE   $$     VAL
                     0000   263         .IF EQ  ?'$$'-^XOEF>
                     0000   264         .IF NDF VAL
                     0000   265         BBSS    S^#VAL,FLAG,.+1
                     0000   266         .IFF
                     0000   267         .IF LT <VAL-8>
                     0000   268         BISB    #<1@VAL>,FLAG
                     0000   269         .IFF
                     0000   270         BBSS    #VAL,FLAG,.+1
                     0000   271         .ENDC
                     0000   272         .ENDC
                     0000   273         .IFF
                     0000   274         BBSS    VAL,FLAG,.+1
                     0000   275         .ENDC
                     0000   276 .ENDM   SETBIT
                     0000   277 ;
```

```
0000    278 ; ======================
0000    279 ;
0000    280 .MACRO  CLRBIT  VAL,FLAG
0000    281         .NTYPE   $$     VAL
0000    282         .IF EQ  2'$$_-^XOEF>
0000    283         .IF NDF VAL
0000    284         BBCC    S^#VAL,FLAG,.+1
0000    285         .IFF
0000    286         .IF LT <VAL-8>
0000    287         BICB    #<1@VAL>,FLAG
0000    288         .IFF
0000    289         BBCC    #VAL,FLAG,.+1
0000    290         .ENDC
0000    291         .ENDC
0000    292         .IFF
0000    293         BBCC    VAL,FLAG,.+1
0000    294         .ENDC
0000    295 .ENDM   CLRBIT
0000    296 ;
0000    297 ; ======================
0000    298 ;
0000    299 .MACRO  INCC    COUNTER,CONTEXT=L,?L    ; Increment counter
0000    300         INC'CONTEXT     COUNTER         ; Do Increment
0000    301         BCC     L                       ; Br if no carry set
0000    302         DEC'CONTEXT     COUNTER         ; Leave at maximum value
0000    303 L:
0000    304 .ENDM   INCC
0000    305 ;
0000    306 ; ======================
0000    307 ;
0000    308 .MACRO  CNTR    CURCNT,COUNTER,CONTEXT=L,?L  ; Accumlate counter
0000    309         ADD'CONTEXT     CURCNT,COUNTER  ; Do addition
0000    310         BCC     L                       ; Br if no carry set
0000    311         MNEG'CONTEXT    #1,COUNTER      ; Leave at maximum value
0000    312 L:
0000    313 .ENDM   CNTR
0000    314 ;
0000    315 ; ======================
0000    316 ;
0000    317 .MACRO  PUSHQ   ARG                     ; Push a quadword
0000    318         MOVQ    ARG,-(SP)               ; Save argument on stack
0000    319 .ENDM   PUSHQ
0000    320 ;
0000    321 ; ======================
0000    322 ;
0000    323 .MACRO  POPQ    ARG                     ; Pop a quadword
0000    324         MOVQ    (SP)+,ARG               ; Restore argument
0000    325 .ENDM   POPQ
0000    326 ;
0000    327 ; ======================
0000    328 ;
0000    329 .MACRO  PARAM   TYPE,OFFSET,WIDTH,MIN,MAX,INVALID,BASE=UCB,STRING,-
0000    330                 SIZE,CHECK=YES
0000    331 ;
0000    332 ;       Macro to generate the parameter tables
0000    333 ;
0000    334 ; Inputs:
```

```
0000    335 ;
0000    336 ;       TYPE = Parameter type
0000    337 ;       OFFSET = Offset in UCB/CDB to current value
0000    338 ;       WIDTH = Width of field in UCB/CDB (B,W,L)
0000    339 ;       MIN = Minimum value parameter is allowed to take
0000    340 ;       MAX = Maximum value parameter is allowed to take
0000    341 ;       INVALID = Invalid flags in status word
0000    342 ;       BASE = Data base (CDB,UCB)
0000    343 ;       STRING = Parameter is a string value
0000    344 ;       SIZE = Maximum size of string parameter in bytes
0000    345 ;       CHECK = Comparison is needed (YES,NO)
0000    346 ;
0000    347         .IF BLANK type
0000    348          .WORD  0
0000    349         .IF_FALSE       ; BLANK type
0000    350          $$$typ = type & prm_typ_m_code         ; Isolate type code
0000    351
0000    352          $$$flg = 0
0000    353          .IIF NOT_BLANK <invalid>, $$$flg = $$$flg!prm_flg_m_invalid
0000    354          .IIF IDN <check><YES>, $$$flg = $$$flg!prm_flg_m_check
0000    355          .IIF IDN <base><CDB>, $$$flg = $$$flg!prm_flg_m_cdb
0000    356
0000    357          .IF BLANK string
0000    358
0000    359                .WORD   $$$typ
0000    360           .IIF NOT_BLANK        <min>, $$$flg = $$$flg!prm_flg_m_min
0000    361           .IIF NOT_BLANK        <max>, $$$flg = $$$flg!prm_flg_m_max
0000    362                .BYTE   $$$flg
0000    363           $$$off = offset & prm_off_m_value     ; Isolate offset only
0000    364           $$$wid = 0                            ; Set null width
0000    365           .IIF IDN <width><B>, $$$wid = <1@prm_off_v_width>
0000    366           .IIF IDN <width><W>, $$$wid = <2@prm_off_v_width>
0000    367           .IIF IDN <width><L>, $$$wid = <3@prm_off_v_width>
0000    368                .WORD   $$$off!$$$wid
0000    369           .IIF NOT_BLANK        <min>, .WORD    min
0000    370           .IIF NOT_BLANK        <max>, .WORD    max
0000    371           line_prm_bufsiz = line_prm_bufsiz + 6
0000    372
0000    373          .IF_FALSE      ; BLANK STRING
0000    374
0000    375                .WORD   $$$typ!prm_typ_m_string ; Indicate a string parameter
0000    376                .BYTE   $$$flg
0000    377            $$$off = offset & prm_off_m_value    ; Isolate offset only
0000    378            $$$wid = <size @ prm_off_v_width> & prm_off_m_width ; Get max allowed
0000    379            $$$siz = <$$$wid @ -prm_off_v_width>
0000    380                .WORD   $$$off!$$$wid
0000    381            line_prm_bufsiz = line_prm_bufsiz + 4 + $$$siz
0000    382
0000    383          .ENDC          ; BLANK STRING
0000    384
0000    385          .IIF NOT_BLANK <invalid>, .WORD    invalid
0000    386
0000    387         .ENDC           ; BLANK TYPE
0000    388 .ENDM   PARAM
0000    389 ;
0000    390 ; ======================
0000    391 ;
```

```
0000    392 .MACRO  OFSET   SIZE,OFFSET,BASE
0000    393         .IF IDN <base><LINE>
0000    394         .WORD   cdb_'size'_'offset'
0000    395         .IFF
0000    396         .WORD   ucb$'size'_XQ_'offset'
0000    397         .ENDC
0000    398 .ENDM   OFSET
0000    399 ;
0000    400 ; ======================
0000    401 ;
0000    402 .MACRO  COUNTER TYPE,WIDTH=16,OFFSET=0,BASE=LINE,BITMAP
0000    403         .IIF NDF 'base'_ctr_size, 'base'_ctr_size = 0
0000    404         .IIF NDF 'base'_ctr_bufsiz, 'base'_ctr_bufsiz = 0
0000    405         .IF IDN <base><LINE>
0000    406         $$$typ = nma$c_ctlin_'type' & nma$m_cnt_typ
0000    407         .IFF
0000    408         $$$typ = nma$c_ctcir_'type' & nma$m_cnt_typ
0000    409         .ENDC
0000    410         $$$wid = 0                          ; Set reserved mask width
0000    411         .IIF IDN <width><8>, $$$wid = <1@nma$v_cnt_wid>
0000    412         .IIF IDN <width><16>, $$$wid = <2@nma$v_cnt_wid>
0000    413         .IIF IDN <width><32>, $$$wid = <3@nma$v_cnt_wid>
0000    414         .IIF EQ $$$wid, .ERROR              ; Invalid bit width value
0000    415         $$$map = 0
0000    416         .IIF IDN <bitmap><MAP>, $$$map = nma$m_cnt_map
0000    417         .WORD   nma$m_cnt_cou!$$$wid!$$$typ!$$$map
0000    418         .IIF IDN <width><8>, OFSET B,'offset,'base
0000    419         .IIF IDN <width><16>, OFSET W,'offset,'base
0000    420         .IIF IDN <width><32>, OFSET L,'offset,'base
0000    421         'base'_ctr_size = 'base'_ctr_size + 1   ; Tally one more entry
0000    422         'base'_ctr_bufsiz = 'base'_ctr_bufsiz + 2 + <width/8>
0000    423         .IIF IDN <bitmap><MAP>, 'base'_ctr_bufsiz = 'base'_ctr_bufsiz + 2
0000    424 .ENDM   COUNTER
0000    425 ;
0000    426 ; ======================
0000    427 ;
0000    428 .MACRO  MOPCTR  WIDTH=16,OFFSET,BITMAP
0000    429         .IIF NDF mop_ctr_size, mop_ctr_size = 0
0000    430         $$$map = 0
0000    431         $$$wid = width/8
0000    432         .IIF IDN <bitmap><MAP>, $$$map = 1@7
0000    433         .IF NOT_BLANK   <offset>
0000    434             .IIF IDN <width><8>, OFSET B,'offset,LINE
0000    435             .IIF IDN <width><16>, OFSET W,'offset,LINE
0000    436             .IIF IDN <width><32>, OFSET L,'offset,LINE
0000    437             .BYTE       $$$map!$$$wid   ; Counter width in bytes + BITMAP FLAG
0000    438             mop_ctr_size = mop_ctr_size + $$$wid
0000    439             .IIF IDN <bitmap><MAP>, mop_ctr_size = mop_ctr_size + 2
0000    440         .IFF ; NOT_BLANK
0000    441             .WORD       0               ; End of table
0000    442         .ENDC ; NOT_BLANK
0000    443 .ENDM   MOPCTR
0000    444 ;
0000    445 ; ======================
0000    446 ;
0000    447 .MACRO  SKIP    BIT,LOC,REG,CONTEXT=W,?L        ; SKIP FIELD
0000    448         BBC     #BIT,LOC,L                  ; Br if field not present
```

```
0000    449          TST'CONTEXT     (REG)+          ; Skip next field
0000    450 1:
0000    451 .ENDM    SKIP
0000    452 ;
0000    453 ; ======================
0000    454 ;
0000    455 .MACRO   $DISPATCH,      INDX,VECTOR,TYPE=W,NMODE=S^#,?MN,?MX,?S,?SS,?ZZ
0000    456 SS:
0000    457
0000    458          .MACRO  $DSP1,$DSP1_1
0000    459                  .IRP    $DSP1_2,$DSP1_1
0000    460                          $DSP2   $DSP1_2
0000    461                  .ENDR
0000    462          .ENDM   $DSP1
0000    463
0000    464          .MACRO  $DSP2,$DSP2_1,$DSP2_2
0000    465                  .=<$DSP2_1-MN>*2 + 5
0000    466                  .WORD   $DSP2_2-S
0000    467          .ENDM   $DSP2
0000    468
0000    469
0000    470          .MACRO  $BND1,$BND1_1,$BND1_2,$BND1_3
0000    471                  $BND2   $BND1_1,$BND1_2
0000    472          .ENDM   $BND1
0000    473
0000    474          .MACRO  $BND2,$BND2_1,$BND2_2
0000    475                  .IIF    $BND2_1,$BND2_2-.,      .=$BND2_2
0000    476          .ENDM   $BND2
0000    477
0000    478          .MACRO  $BND    $BND_1,$BND_2
0000    479                  .IRP    $BND_3,<$BND_2>
0000    480                          $BNDT   $BND_1,$BND_3
0000    481                  .ENDR
0000    482          .ENDM   $BND
0000    483
0000    484          .=0
0000    485 ZZ:
0000    486          $BND    GT,<VECTOR>
0000    487 MX:
0000    488          $BND    LT,<VECTOR>
0000    489 MN:
0000    490          .=SS
0000    491
0000    492 CASE'TYPE        INDX,#<MN-ZZ>,NMODE'<MX-MN>
0000    493 S:
0000    494          .REPT   MX-MN+1
0000    495          .WORD   <MX-MN>*2 + 2
0000    496          .ENDR
0000    497
0000    498          .=S
0000    499
0000    500          $DSP1   <<VECTOR>>
0000    501
0000    502          .=<MX-MN>*2 + S + 2
0000    503
0000    504 .ENDM    $DISPATCH
0000    505
```

```
                     0000   507 ;
                     0000   508 ; Overlays of IRP
                     0000   509 ;
                     0000   510         $DEFINI IRP     GLOBAL
                     0000   511
00000021             0000   512 . = IRP$W_FUNC+1
                     0021   513 $DEF     IRP$B_XQ_FUNC   .BLKB   1       ; QNA driver internal function code
                     0022   514
00000038             0022   515 . = IRP$L_MEDIA
                     0038   516 $DEF     IRP$W_XQ_RID                    ; RCV/XMT request ID
                     0038   517 $DEF     IRP$B_XQ_SLOT   .BLKB   1       ; RCV/XMT mapping slot number
                     0039   518 $DEF     IRP$B_XQ_RING   .BLKB   1       ; RCV/XMT ring entry number
                     003A   519
00000046             003A   520 . = IRP$Q_STATION+6
                     0046   521 $DEF     IRP$B_XQ_DATAP  .BLKB   1       ; XMT buffered data path number
                     0047   522
0000003C             0047   523 . = IRP$L_MEDIA+4
                     003C   524 $DEF     IRP$L_XQ_SYSBUF .BLKL   1       ; XMT system buffer address
                     0040   525
0000003C             0040   526 . = IRP$L_MEDIA+4
                     003C   527 $DEF     IRP$L_XQ_DATBUF .BLKL   1       ; User RCV data buffer address
                     0040   528
00000038             0040   529 . = IRP$L_MEDIA
                     0038   530 $DEF     IRP$W_XQ_USERSIZ .BLKW  1       ; User P2 buffer size on sensemode
                     003A   531
0000003A             003A   532 . = IRP$L_MEDIA+2
                     003A   533 $DEF     IRP$W_XQ_STATUS .BLKW   1       ; Completion status
                     003C   534
0000003C             003C   535 . = IRP$L_MEDIA+4
                     003C   536 $DEF     IRP$L_XQ_USERBUF .BLKL  1       ; User P1 buffer address on sensemode
                     0040   537
00000040             0040   538 . = IRP$Q_STATION
                     0040   539 $DEF     IRP$L_XQ_P2BUF  .BLKL   1       ; User P2 buffer address on sensemode
                     0044   540
00000044             0044   541 . = IRP$Q_STATION+4
                     0044   542 $DEF     IRP$W_XQ_P2SIZ  .BLKW   1       ; P2 return buffer size on sensemode
                     0046   543
00000040             0046   544 . = IRP$Q_STATION
                     0040   545 $DEF     IRP$W_XQ_CODE   .BLKW   1       ; Bad parameter code on startup request
                     0042   546
0000003C             0042   547 . = IRP$L_MEDIA+4
                     003C   548 $DEF     IRP$L_XQ_MAP    .BLKL   1       ; Diagnostics buffer mapping info
                     0040   549
00000040             0040   550 . = IRP$Q_STATION
                     0040   551 $DEF     IRP$L_XQ_DGUNI  .BLKL   1       ; Diagnostics buffer UNIBUS address
                     0044   552 $DEF     IRP$L_XQ_UPADR  .BLKW   1       ; Micro-process internal address
                     0046   553
00000094             0046   554 . = IRP$L_RBUFH_AD
                     0094   555 $DEF     IRP$L_XQ_SETUP  .BLKL   1       ; Setup transmit buffer
                     0098   556
0000003A             0098   557 . = IRP$L_MEDIA+2
                     003A   558 $DEF     IRP$W_XQ_PROTYP .BLKW   1       ; Protocol type for user
                     003C   559
00000090             003C   560 . = IRP$L_LBOFF
                     0090   561 $DEF     IRP$L_XQ_SHR    .BLKL   1       ; Address of SHR structure for user
                     0094   562
00000060             0094   563 . = IRP$L_FQFL
```

```
0060    564 $DEF    IRP$C_XQ_STD    .BLKL   1       ; End of "standard" IRP
0064    565
0064    566 ;
0064    567 ; Define driver internal function codes stored in IRP$B_XQ_FUNC of IRP.
0064    568 ; NOTE: These are not really used as bit offsets - but as values.
0064    569 ;
0064    570         _VIELD  XQ_FC,0,<-              ; Internal function codes
0064    571                 <INIT>,-                ; Initialize QNA
0064    572                 <XMIT>,-                ; Transmit request
0064    573                 <RECV>,-                ; Receive request
0064    574                 <STOP>,-                ; Stop protocol
0064    575                 <CANCEL>,-              ; Cancel request
0064    576                 <RESTART>,-             ; Restart PROTOCOL
0064    577                 <CHMODE>,-              ; Change the setup mode
0064    578                 >                       ;
0064    579
0064    580         $DEFEND IRP                     ; End of IRP overlays
```

```
                     0000   582 ;
                     0000   583 ; Overlays of CXB
                     0000   584 ;
                     0000   585         $DEFINI CXB     GLOBAL
                     0000   586
00000020             0000   587 . = CXB$L_SPARE1
                     0000   588 $DEF     CXB$B_XQ_FUNC   .BLKB   1       ; QNA driver internal function code
00000022             0001   589                         .BLKB   1       ; SPARE
                     0002   590 $DEF     CXB$W_XQ_RID                    ; RCV/XMT request ID
                     0002   591 $DEF     CXB$B_XQ_SLOT   .BLKB   1       ; RCV/XMT mapping slot number
                     0003   592 $DEF     CXB$B_XQ_RING   .BLKB   1       ; RCV/XMT ring entry number
                     0004   593
                     0004   594 ;
                     0004   595 ; The following overlays are for transmits only
                     0004   596 ;
00000024             0004   597 . = CXB$L_SPARE0
                     0004   598         ;
                     0004   599         ;  NOTE: The following two fields area overlayed. So if the Low
                     0004   600         ;  Bit is set, then the address is that of a UCB, else it's an IRP.
                     0004   601         ;
                     0004   602 $DEF     CXB$L_T_IRP                     ; Associated IRP address
                     0004   603 $DEF     CXB$L_T_UCB     .BLKL   1       ; Associated UCB address
                     0008   604
0000003A             0008   605 . = CXB$C_HEADER-<XQ_C_HEADER>
                     003A   606 $DEF     CXB$T_T_DATA    .BLKB   XQ_C_HEADER ; Standard Ethernet header
                     0048   607
                     0048   608 ;
                     0048   609 ; The following overlays are for receives only
                     0048   610 ;
0000000B             0048   611 . = CXB$B_CODE
                     000B   612 $DEF     CXB$B_R_FLAGS   .BLKB   1       ; Receive message flags
                     000C   613
0000001C             000C   614 . = CXB$L_END_ACTION
                     001C   615 $DEF     CXB$W_R_NCHAIN  .BLKW   1       ; Number of buffers in chain
                     001E   616
00000014             001E   617 . = CXB$L_IRP
                     0014   618 $DEF     CXB$W_R_STS     .BLKW   1       ; Receive status
                     0016   619
00000038             0016   620 . = CXB$C_HEADER - <XQ_C_HEADER+XQ_C_CNTSIZ>
                     0038   621 $DEF     CXB$T_R_DATA                    ; Start of receive data
                     0038   622 $DEF     CXB$G_R_DEST    .BLKW   3       ; Destination node address
                     003E   623 $DEF     CXB$G_R_SRC     .BLKW   3       ; Source node address
                     0044   624 $DEF     CXB$W_R_PTYPE   .BLKW   1       ; Protocol Type
                     0046   625 $DEF     CXB$T_R_USERDAT                 ; Start of user data
                     0046   626 $DEF     CXB$W_R_SIZE    .BLKW   1       ; Size of received message (if padded).
                     0048   627
                     0048   628 ;
                     0048   629 ; NOTE: The CXB functions are the same as for an IRP (IRP$B_XQ_FUNC)
                     0048   630 ;
                     0048   631
                     0048   632         $DEFEND CXB                     ; End of CXB overlays
```

```
                     0000   634 ;
                     0000   635 ; Definitions that follow the standard UCB fields
                     0000   636 ;
                     0000   637
                     0000   638          $DEFINI UCB     GLOBAL                  ; Start of UCB definitions
                     0000   639
00000098             0000   640 . = UCB$C_NI_LENGTH                              ; Position at end of UCB NI extension
                     0098   641
                     0098   642 $DEF     UCB$Q_XQ_QUEUES                         ; Message and I/O request queue heads
                     0098   643 $DEF     UCB$Q_XQ_SHARE  .BLKQ   1               ; List of shared users
                     00A0   644 $DEF     UCB$Q_XQ_IOQS                           ; Start of the I/O queues
                     00A0   645 $DEF     UCB$Q_XQ_RCVMSG .BLKQ   1               ; Receive messages completed
                     00A8   646 $DEF     UCB$Q_XQ_RCVREQ .BLKQ   1               ; Receive IRP waiting for messages
                     00B0   647 $DEF     UCB$Q_XQ_XMTREQ .BLKQ   1               ;% Xmit IRP wait queue (PT-TO-PT)
00000004             00B8   648 UCB$C_XQ_QUEUES = <.-UCB$Q_XQ_QUEUES>/8 ; Number of queue heads
                     00B8   649
                     00B8   650 $DEF     UCB$L_XQ_PID    .BLKL   1               ; Starter's PID
                     00BC   651 $DEF     UCB$L_XQ_CPID   .BLKL   1               ; Creator's PID
                     00C0   652 $DEF     UCB$L_XQ_AST    .BLKL   1               ; Attention AST list
                     00C4   653 $DEF     UCB$L_XQ_DEFUSR .BLKL   1               ; Default shared user (shared use only)
;RNG0001              00C8     .1 $DEF   UCB$L_XQ_CDB    .BLKL   1               ; Address of CDB
                     00CC   654 $DEF     UCB$W_XQ_QUOTA  .BLKW   1               ; Receive buffer quota
                     00CE   655 $DEF     UCB$W_XQ_PROTYP .BLKW   1               ; Ethernet protocol type
                     00D0   656
                     00D0   657 $DEF     UCB$B_XQ_SETPRM                         ; Start of parameter section
                     00D0   658 $DEF     UCB$G_XQ_DES    .BLKW   3               ; Destination address for shared user
                     00D6   659 $DEF     UCB$W_XQ_HBQ    .BLKW   1               ; Hardware buffer quota
                     00D8   660 $DEF     UCB$B_XQ_ACC    .BLKB   1               ; Protocol access mode
                     00D9   661 $DEF     UCB$B_XQ_BFN    .BLKB   1               ; Number of receive buffers
                     00DA   662
                     00DA   663 $DEF     UCB$B_XQ_SHRPRM                         ; Start of "shared user" validated prms
                     00DA   664 $DEF     UCB$W_XQ_BSZ    .BLKW   1               ; Device buffer size
                     00DC   665 $DEF     UCB$B_XQ_PRO    .BLKB   1               ; Protocol selection
                     00DD   666 $DEF     UCB$B_XQ_PAD    .BLKB   1               ; Padding mode
                     00DE   667 $DEF     UCB$B_XQ_PRM    .BLKB   1               ; Promiscuous mode
                     00DF   668 $DEF     UCB$B_XQ_MLT    .BLKB   1               ; Multicast (all) address state
                     00E0   669 $DEF     UCB$B_XQ_DCH    .BLKB   1               ; Data chaining on receives
                     00E1   670
                     00E1   671 $DEF     UCB$B_XQ_CDBPRM                         ; Start of settable parameters for CDB
                     00E1   672 $DEF     UCB$B_XQ_CON    .BLKB   1               ; Controller mode
00000001             00E2   673 UCB$C_XQ_CDBPRM = .-UCB$B_XQ_CDBPRM
00000008             00E2   674 UCB$C_XQ_SHRPRM = .-UCB$B_XQ_SHRPRM
                     00E2   675
                     00E2   676 $DEF     UCB$G_XQ_PHA    .BLKW   3               ; User defined physical address
00000007             00E8   677 UCB$C_XQ_SETPRM = .-UCB$B_XQ_CDBPRM
                     00E8   678
                     00E8   679 $DEF     UCB$B_XQ_MST    .BLKB   1               ; Maintenance state
                     00E9   680 $DEF     UCB$B_XQ_MULTI  .BLKB   1               ; Number of entries in MULTI
                     00EA   681 $DEF     UCB$B_XQ_MLTTBL .BLKB   1               ; Number of entries in MLTTBL
;RNG0001              00EB     .1 $DEF   UCB$B_XQ_SPARE  .BLKB   1               ; SPARE BYTE
                     00EC   682 $DEF     UCB$G_XQ_MULTI  .BLKW   3*MAX_C_MLT ; Multicast address list
                     0134   683 $DEF     UCB$G_XQ_MLTTBL .BLKW   3*MAX_C_MLT ; Multicast generation list
                     017C   684
                     017C   685 $DEF     UCB$W_XQ_CTR                            ; Start of counter section
                     017C   686 $DEF     UCB$W_XQ_MNECTR .BLKW   1               ; Multicast address not enabled
                     017E   687 $DEF     UCB$W_XQ_UBUCTR .BLKW   1               ; No buffer available counter
                     0180   688 $DEF     UCB$L_XQ_SBLCTR .BLKL   1               ; Number of blocks sent
```

```
0184    689 $DEF    UCB$L_XQ_SBYCTR .BLKL   1       ; Number of bytes sent
0188    690 $DEF    UCB$L_XQ_RBLCTR .BLKL   1       ; Number of blocks received
018C    691 $DEF    UCB$L_XQ_RBYCTR .BLKL   1       ; Number of bytes received
0190    692                                         ; Unused/unneeded fields
0190    693 $DEF    UCB$W_XQ_TOTQUO .BLKW   1       ; Total quota for shared UCB
0192    694 $DEF    UCB$L_XQ_FFI    .BLKL   1       ; Fast interface BLOCK address
0196    695 $DEF    UCB$L_XQ_STIRP  .BLKL   1       ;% Address of PT-TO-PT Startup IRP
019A    696
019A    697 $DEF    UCB$C_XQ_LENGTH                 ; Size of XQDRIVER UCB
019A    698
019A    699 ;
019A    700 ; Define device status bits
019A    701 ;
019A    702         $VIELD  UCB,0,<-                ; XQDRIVER UCB$W_DEVSTS bits
019A    703                 <XQ_INITED,,M>,-        ; Device is initialized
019A    704                 <,15,-                  ; RESERVED
019A    705                 <XQ_PROTYP,,M>,-        ; Protocol type specified
019A    706                 <XQ_SHARE,,M>,-         ; Shared protocol type
019A    707                 <XQ_RUN,,M>,-           ; Unit is in RUN mode
019A    708                 <XQ_START,,M>,-         ;% Unit is in PT-TO-PT startup state
019A    709                 <XQ_STACK,,M>,-         ;% Unit is in PT-TO-PT stack state
019A    710                 <,7>,-                  ; Reserved
019A    711                 <XQ_INTERLOCK,,M>,-     ; RESTART bit is interlocked
019A    712                 <XQ_RESTART,,M>,-       ; Automatic RESTART on PROTOCOL
019A    713                 >                       ;  requested
019A    714
019A    715         $DEFEND UCB                     ; End of UCB definitions
```

```
0000    717 ;
0000    718 ; Device register offsets and bit definitions
0000    719 ;
0000    720         $DEFINI XQ      GLOBAL          ; Start of port CSR definitions
0000    721
0000    722 $DEF    PHYADD0 .BLKW   1               ; Physical address (R/O) - low byte
0002    723 $DEF    PHYADD1 .BLKW   1               ;   more physical address (R/O)
0004    724 $DEF    PHYADD2                         ;   and still more (R/O)
0004    725 $DEF    RCVLIST .BLKW   1               ; Receive descriptor list (W/O)
0006    726 $DEF    RCVLST1                         ;   high order receive list (W/O)
0006    727 $DEF    PHYADD3 .BLKW   1               ;   more physical address (R/O)
0008    728 $DEF    PHYADD4                         ;   more physical address (R/O)
0008    729 $DEF    XMTLIST .BLKW   1               ; Transmit descriptor list (W/O)
000A    730 $DEF    XMTLST1                         ;   high order receive list (W/O)
000A    731 $DEF    PHYADD5 .BLKW   1               ;   more physical address (R/O)
000C    732 $DEF    VECTOR  .BLKW   1               ; Vector addrss (R/W)
000E    733 $DEF    CSR     .BLKW   1               ; Port CSR
0010    734
0010    735         _VIELD  XQ_CSR,0,<-             ; CSR bit definitions
0010    736                 <RCVENA,,M>,-           ; Receive Enable
0010    737                 <RESET,,M>,-            ; Reset
0010    738                 <NXM,,M>,-              ; Non-existent memory
0010    739                 <RROM,,M>,-             ; Read BOOT/DIAGNOSTICS ROM
0010    740                 <XMTINV,,M>,-           ; Transmit list is invalid
0010    741                 <RCVINV,,M>,-           ; Receive list is invalid
0010    742                 <INTENA,,M>,-           ; Interrupt enable
0010    743                 <RCVINT,,M>,-           ; Receive interrupt
0010    744                 <ILOOP,,M>,-            ; Internal loopback (0=ENABLE,1=DISABLE)
0010    745                 <ELOOP,,M>,-            ; External loopback
0010    746                 <SANITY,,M>,-           ; Sanity timer
0010    747                 <,1>,-                  ; RESERVED
0010    748                 <XCAB,,M>,-             ; Transceiver cable okay
0010    749                 <CAR,,M>,-              ; Carrier sense
0010    750                 <ERR,,M>,-              ; Fatal error flag (software set)
0010    751                 <XMTINT,,M>,-           ; Transmit interrupt
0010    752                 >
0010    753
0010    754         _VIELD  XQ_SOFT,0,<-            ; Software error flag bit definitions
0010    755                 <TIMEOUT,,M>,-          ; Timeout
0010    756                 <POWER,,M>,-            ; Powerfail
0010    757                 >
0010    758
0010    759         $DEFEND XQ                      ; End of device register definitions
```

```
                     0000   761
                     0000   762 ;
                     0000   763 ; Define the Transmit Ring Entry
                     0000   764 ;
                     0000   765          $DEFINI XMT     GLOBAL          ; Start of Transmit Ring Entry format
                     0000   766
                     0000   767 $DEF     XMT_W_FLAG      .BLKW   1       ; Flags word
                     0002   768 $DEF     XMT_W_ADDRHI    .BLKW   1       ; Buffer address (high) and descriptor
                     0004   769 $DEF     XMT_W_ADDR      .BLKW   1       ; Buffer address (low 16 bits)
                     0006   770 $DEF     XMT_W_LEN       .BLKW   1       ; 2's complement WORD size
                     0008   771 $DEF     XMT_W_STS       .BLKW   1       ; XMIT Status word
                     000A   772 $DEF     XMT_W_TDR       .BLKW   1       ; Time Domain Reflectometry word
                     000C   773 $DEF     XMT_C_LENGTH                    ; Size of transmit buffer ring entry
0000003C             000C   774 XMT_K_LENGTH = XMT_C_LENGTH * <MAX_C_XMT+1> ; Size of xmit ring (1 for chain)
                     000C   775
                     000C   776          _VIELD  XMT_FLG,0,<-            ; Define flag bits
                     000C   777                  <,14>,-                 ; RESERVED
                     000C   778                  <ERR,,M>,-              ; Transmit error
                     000C   779                  <LAST,,M>,-             ; LAST packet/NOT used indicator
                     000C   780                                         -;  The driver only queues one segment
                     000C   781                                         -;  transmit buffers, so this is
                     000C   782                                         -;  essentially an OWN flag.
                     000C   783                  >
                     000C   784
                     000C   785          _VIELD  XMT_DSC,0,<-            ; Define bits for descriptor word
                     000C   786                  <,65>,-                 ; RESERVED for high order address
                     000C   787                  <BEGODD,1,M>,-          ; Buffer begins on a ODD address
                     000C   788                  <ENDODD,1,M>,-          ; Buffer ends on an ODD address
                     000C   789                  <,4>,-                  ; RESERVED
                     000C   790                  <SETUP,1,M>,-           ; Setup operation
                     000C   791                  <EOM,1,M>,-             ; End of message.
                     000C   792                  <CHAIN,1,M>,-           ; Chain operation
                     000C   793                  <VALID,1,M>,-           ; Valid buffer address
                     000C   794                  >
                     000C   795
                     000C   796          _VIELD  XMT_STS,0,<-            ; Define bits for status word
                     000C   797                  <,45>,-                 ; RESERVED
                     000C   798                  <COL,4,M>,-             ; Number of collisions
                     000C   799                  <FAIL,,>,-              ; Collision check failure
                     000C   800                  <ABORT,,>,-             ; Transmission was aborted due to
                     000C   801                                         -;  excessive collisions
                     000C   802                  <,1>,-                  ; RESERVED
                     000C   803                  <NOCAR,,M>,-            ; No carrier ever present
                     000C   804                  <LCAR,,M>,-             ; Loss of carrier
                     000C   805                  <,1>,-                  ; RESERVED
                     000C   806                  <ERR,,M>,-              ; Error on transmit
                     000C   807                  <LAST,,M>,-             ; LAST packet/NOT used indicator
                     000C   808                                         -;  The driver only queues one segment
                     000C   809                                         -;  transmit buffers, so this is
                     000C   810                                         -;  essentially an OWN flag.
                     000C   811                  >
                     000C   812
                     000C   813          _VIELD  XMT_TDR,0,<-            ; Define bits for TDR WORD
                     000C   814                  <TDR,14,M>,-            ; Time Domain Reflectometry
                     000C   815                  <,2>,-                  ; RESERVED
                     000C   816                  >
                     000C   817
```

```
                     000C   818          $DEFEND XMT                        ; End of Transmit Ring Entry
                     0000   819 ;
                     0000   820 ; Define the Receive List Entry
                     0000   821 ;
                     0000   822          $DEFINI RCV     GLOBAL             ; Start of Receive List Entry format
                     0000   823
                     0000   824 $DEF     RCV_W_FLAG      .BLKW   1          ; Flags word
                     0002   825 $DEF     RCV_W_ADDRHI    .BLKW   1          ; Buffer address (high) and descriptor
                     0004   826 $DEF     RCV_W_ADDR      .BLKW   1          ; Buffer address (low 16 bits)
                     0006   827 $DEF     RCV_W_LEN       .BLKW   1          ; 2's complement WORD size
                     0008   828 $DEF     RCV_W_STS       .BLKW   1          ; Status word
                     000A   829 $DEF     RCV_W_LENB      .BLKW   1          ; Receive length byte <7:0>
                     000C   830 $DEF     RCV_C_LENGTH                       ; Size of transmit buffer ring entry
          0000006C   000C   831 RCV_K_LENGTH = RCV_C_LENGTH * <MAX_C_RCV+1> ; Size of receive ring (1 for chain)
                     000C   832
                     000C   833          _VIELD  RCV_FLG,0,<-               ; Define flag bits
                     000C   834                  <,14>,-                    ; RESERVED
                     000C   835                  <ERR,,M>,-                 ; Receive error
                     000C   836                  <LAST,,M>,-                ; LAST packet/NOT used indicator
                     000C   837                  >
                     000C   838
                     000C   839          _VIELD  RCV_DSC,0,<-               ; Define bits for descriptor word
                     000C   840                  <,14>,-                    ; RESERVED for high order address
                     000C   841                  <CHAIN,1,M>,-              ; Chain operation
                     000C   842                  <VALID,1,M>,-              ; Valid buffer address
                     000C   843                  >
                     000C   844
                     000C   845          _VIELD  RCV_STS,0,<-               ; Define status word
                     000C   846                  <OVF,,M>,-                 ; DEQNA receive overflow
                     000C   847                  <CRCERR,,M>,-              ; CRC error
                     000C   848                  <FRAME,,M>,-               ; Framing error
                     000C   849                  <SHORT,,M>,-               ; Short on Ethernet Cable
                     000C   850                  <,4>,-                     ; RESERVED
                     000C   851                  <RLEN,3,M>,-               ; Receive length bits <10:8>
                     000C   852                  <RUNT,,M>,-                ; RUNT packet
                     000C   853                  <DISCARD,,M>,-             ; DISCARD packet (VALIDATES OVF & CRCERR)
                     000C   854                  <ESETUP,,M>,-              ; End of setup
                     000C   855                  <ERR,,M>,-                 ; Error/USED indicator
                     000C   856                  <LAST,,M>,-                ; LAST packet/NOT used indicator
                     000C   857                  >
                     000C   858
                     000C   859          $DEFEND RCV                        ; End of Receive Ring Entry
                     0000   860
                     0000   861 ;
                     0000   862 ; Transmit Buffer Header Format
                     0000   863 ;
                     0000   864          $DEFINI XBUF                       ; Define transmit buffer header
                     0000   865
                     0000   866 $DEF     XBUF_G_DEST     .BLKW   3          ; Destination address
                     0006   867 $DEF     XBUF_G_SRC      .BLKW   3          ; Source address (overlays UCB)
                     000C   868 $DEF     XBUF_W_TYPE     .BLKW   1          ; Protocol type
                     000E   869 $DEF     XBUF_T_DATA                        ; Start of xmit data
                     000E   870 $DEF     XBUF_C_HEADER                      ; Size of buffer header
                     000E   871
                     000E   872 $DEF     XBUF_W_SIZE     .BLKW   1          ; Size of buffer (only if padding)
                     0010   873
                     0010   874          $DEFEND XBUF                       ; End of transmit buffer header
```

```
                        ;
                        ; Block header for non-DECnet xmit buffers
                        ;
                        $DEFINI BLK                             ; Define a standard block header
                        
$DEF                    BLK_L_LINK      .BLKL   2               ; Forward and backward queue links
$DEF                    BLK_W_SIZE      .BLKW   1               ; Block size
$DEF                    BLK_B_TYPE      .BLKB   1               ; Block type
$DEF                    BLK_B_SPARE     .BLKB   1               ; SPARE byte
$DEF                    BLK_T_DATA                              ; Start of data
$DEF                    BLK_C_HEADER                            ; Size of buffer header
                        
                        $DEFEND BLK
```

```
                     0000   890 ;
                     0000   891 ; Define the DEQNA Controller Data Block (CDB) fields
                     0000   892 ;
                     0000   893
                     0000   894          $DEFINI CDB     GLOBAL          ; Start of CDB definitions
                     0000   895
                     0000   896 $DEF     CDB_L_FQFL      .BLKL   1       ; Fork queue forward link
                     0004   897 $DEF     CDB_L_FQBL      .BLKL   1       ; Fork queue backward link
                     0008   898 $DEF     CDB_W_SIZE      .BLKW   1       ; Size of CDB
                     000A   899 $DEF     CDB_B_TYPE      .BLKB   1       ; Type of structure
                     000B   900 $DEF     CDB_B_FIPL      .BLKB   1       ; Fork IPL
                     000C   901 $DEF     CDB_L_FPC       .BLKL   1       ; Fork PC
;RNG0001              0010    .1 $DEF     CDB_L_LASTCSR                   ; Port CSR contents
-1                   0010   903 $DEF     CDB_L_FR3       .BLKL   1       ; Fork R3
                     0014   904 $DEF     CDB_L_FR4       .BLKL   1       ; Fork R4
                     0018   905 $DEF     CDB_B_NEXTXMT   .BLKB   1       ; Next entry in XMT ring
                     0019   906 $DEF     CDB_B_NEXTRCV   .BLKB   1       ; Next entry in RCV ring
                     001A   907 ASSUME   MAX_C_XMT LE 8
                     001A   908 ASSUME   MAX_C_RCV LE 8
                     001A   909 $DEF     CDB_B_RCVMAP    .BLKB   1       ; RCV map slot in use flags
                     0018   910 $DEF     CDB_B_XMTMAP    .BLKB   1       ; XMT map slot in use flags
                     001C   911 $DEF     CDB_L_RCVMAP    .BLKL   MAX_C_RCV-1 ; RCV mapping vector
                     0038   912 $DEF     CDB_L_XMTMAP    .BLKL   MAX_C_XMT-1 ; XMT mapping vector
                     0044   913 $DEF     CDB_L_RRINGPA   .BLKL   MAX_C_RCV+1 ; RCV Ring entry PHYSICAL address
                     0068   914 $DEF     CDB_L_XRINGPA   .BLKL   MAX_C_XMT+1 ; XMT Ring entry PHYSICAL address
                     007C   915 $DEF     CDB_L_RRINGVA   .BLKL   MAX_C_RCV ; RCV Ring entry VIRTUAL address
                     009C   916 $DEF     CDB_L_XRINGVA   .BLKL   MAX_C_XMT ; XMT Ring entry VIRTUAL address
                     00AC   917 $DEF     CDB_L_RCV_PA    .BLKL   MAX_C_RCVUV1 ; Receive contiguous buffer
                     00C0   918                                          ;  physical address
                     00C0   919 $DEF     CDB_L_XMT_PA    .BLKL   MAX_C_XMTUV1 ; Transmit contiguous buffer
                     00C4   920                                          ;  physical address
                     00C4   921 $DEF     CDB_L_RCV_VA    .BLKL   MAX_C_RCVUV1 ; Receive contiguous buffer
                     00D8   922                                          ;  virtual address
                     00D8   923 $DEF     CDB_L_XMT_VA    .BLKL   MAX_C_XMTUV1 ; Transmit contiguous buffer
                     00DC   924                                          ;  virtual address
                     00DC   925 $DEF     CDB_Q_QUEUES                    ; Start of CDB queues
                     00DC   926 $DEF     CDB_Q_XMTREQ    .BLKQ   1       ; Transmit request queue
                00000001    00E4   927 CDB_C_ABORTS = <.-CDB_Q_QUEUES>/8 ; Number of Queues to abort requests
-1                   00E4   929 $DEF     CDB_Q_XMTPND    .BLKQ   1       ; Transmit pending queue
                     00EC   930 $DEF     CDB_Q_RCVBUF    .BLKQ   1       ; Receive buffer queue
                     00F4   931 $DEF     CDB_Q_RCVPND    .BLKQ   1       ; Receive pending queue
                     00FC   932 $DEF     CDB_Q_POST      .BLKQ   1       ; Post process queue
                00000005    0104   933 CDB_C_QUEUES = <.-CDB_Q_QUEUES>/8 ; Number of Queue Heads
                     0104   934
                     0104   935 $DEF     CDB_B_LASTRCV   .BLKB   1       ; Last entry done in RCV ring
                     0105   936 $DEF     CDB_B_LASTXMT   .BLKB   1       ; Last entry done in XMT ring
                     0106   937 $DEF     CDB_B_RCVCNT    .BLKB   1       ; Count of receives given to QNA
                     0107   938 $DEF     CDB_B_XMTCNT    .BLKB   1       ; Count of xmits given to QNA
                     0108   939 $DEF     CDB_W_BSZ       .BLKW   1       ; Device buffer size
                     010A   940 $DEF     CDB_W_QUOTA     .BLKW   1       ; SYSTEM buffer quota
                     010C   941
                     010C   942 $DEF     CDB_L_DEVDEPEND .BLKL   1       ; Device dependent longword
                     0110   943 $DEF     CDB_L_UCB0      .BLKL   1       ; Address of UCB #0
;RNG0001              0114    .1 $DEF     CDB_L_CSR       .BLKL   1       ; DEQNA CSR address
;RNG0001              0118    .2 $DEF     CDB_B_SPARE     .BLKB   1       ; SPARE BYTE
;RNG0001              0119    .3 $DEF     CDB_B_DIAG1     .BLKB   1       ; Diagnostic info byte
;RNG0001              011A    .4 $DEF     CDB_W_DIAG2     .BLKW   1       ; Second word of diagnostic info
```

```
;RNG0001                011C     .5
;RNG0001                011C     .6 $DEF     CDB_L_RINGMAP   .BLKL   1       ; Mapping information for RINGs
-3                      0120   947
                        0120   948 $DEF     CDB_G_COUNTER                   ; Start of counters
                        0120   949 $DEF     CDB_W_ZERO      .BLKW   1       ; Seconds since last zeroed
                        0122   950 $DEF     CDB_L_DBRCTR    .BLKL   1       ; Messages received
                        0126   951 $DEF     CDB_L_MBLCTR    .BLKL   1       ; Multicast messages received
                        012A   952 $DEF     CDB_W_RFLMAP    .BLKW   1       ; Messages received in error - bitmap
                        012C   953 $DEF     CDB_W_RFLCTR    .BLKW   1       ; Messages received in error
                        012E   954         ASSUME  CDB_W_RFLCTR EQ CDB_W_RFLMAP+2
                        012E   955 $DEF     CDB_L_BRCCTR    .BLKL   1       ; Bytes received
                        0132   956 $DEF     CDB_L_MBYCTR    .BLKL   1       ; Multicast bytes received
                        0136   957 $DEF     CDB_W_OVRCTR    .BLKW   1       ; Packets lost due to hardware buffers
                        0138   958 $DEF     CDB_W_LBECTR    .BLKW   1       ; Packets lost due to system buffer error
                        013A   959 $DEF     CDB_L_DBSCTR    .BLKL   1       ; Messages sent
                        013E   960 $DEF     CDB_L_MBSCTR    .BLKL   1       ; Multicast messages sent
                        0142   961 $DEF     CDB_L_BSMCTR    .BLKL   1       ; Messages sent - several errors
                        0146   962 $DEF     CDB_L_BS1CTR    .BLKL   1       ; Messages sent - 1 error
                        014A   963 $DEF     CDB_L_BIDCTR    .BLKL   1       ; Messages sent - initially deferred
                        014E   964 $DEF     CDB_L_BSNCTR    .BLKL   1       ; Bytes sent
                        0152   965 $DEF     CDB_L_MSNCTR    .BLKL   1       ; Multicast bytes sent
                        0156   966 $DEF     CDB_W_SFLMAP    .BLKW   1       ; Send failures - bitmap
                        0158   967 $DEF     CDB_W_SFLCTR    .BLKW   1       ; Send failures
                        015A   968         ASSUME  CDB_W_SFLCTR EQ CDB_W_SFLMAP+2
                        015A   969 $DEF     CDB_W_CDCCTR    .BLKW   1       ; Transmit collision check fa lure
                        015C   970
                        015C   971 $DEF     CDB_W_UFDCTR    .BLKW   1       ; No protocol type counter on receive
                        015E   972 $DEF     CDB_W_SBUCTR    .BLKW   1       ; System buffer unavailable
                        0160   973 $DEF     CDB_W_UBUCTR    .BLKW   1       ; No buffer available on all PTs
                        0162   974
-1                      0162   976 ;
                        0162   977 ; Define the UNIBUS mapped portion of the CDB (QNA data structures)
                        0162   978 ;
                        0162   979 $DEF     CDB_G_MAPPED                    ; Start of UNIBUS mapped portion of CDB
                        0162   980 $DEF     CDB_G_RRING     .BLKB   RCV_K_LENGTH    ; Recv ring buffer
                        01CE   981 $DEF     CDB_G_XRING     .BLKB   XMT_K_LENGTH    ; Xmit ring buffer
                 000000A8   020A   982 CDB_C_MAPPED = .-CDB_G_MAPPED            ; Size of UNIBUS mapped portion of CDB
                        020A   983 $DEF     CDB_C_ZERO                      ; Size of CDB to zero (everything from
                        020A   984                                          ;  the beginning to this point).
                        020A   985 $DEF     CDB_B_TIM_XMT   .BLKB   1       ; Transmit timer cell
                        020B   986 $DEF     CDB_B_UNTCNT    .BLKB   1       ; Number of active units (UCBs)
;RNG0001                020C     .1 $DEF     CDB_L_UV1BUF    .BLKL   1       ; MicroVAX I buffer area address
-1                      0210   988 $DEF     CDB_L_PRMUSER   .BLKL   1       ; Promiscuous user's UCB address
                        0214   989 $DEF     CDB_L_TQE       .BLKB   TQE$C_LENGTH ; Timer queue elem- t
                        0244   990 $DEF     CDB_W_MODE      .BLKW   1       ; QNA hardware mode
                        0246   991 $DEF     CDB_B_STS       .BLKB   1       ; Controller status
                        0247   992 $DEF     CDB_B_PRM       .BLKB   1       ; Promiscuous mode
                        0248   993 $DEF     CDB_B_MLT       .BLKB   1       ; Multicast (all) address state
                        0249   994
                        0249   995 $DEF     CDB_B_SETPRM                    ; Start of settable parameters
                        0249   996 $DEF     CDB_B_CON       .BLKB   1       ; Controller mode
                 00000001   024A   997 CDB_C_SETPRM = .-CDB_B_SETPRM          ; Size of settable parameter list
                        024A   998
                        024A   999 $DEF     CDB_G_PHA       .BLKW   3       ; User defined physical address
                        0250  1000 $DEF     CDB_G_HWA       .BLKW   3       ; Hardware physical address
                        0256  1001 $DEF     CDB_G_PHYADR    .BLKW   3       ; The current hardware address
                        025C  1002 $DEF     CDB_B_MULTI     .BLKB   1       ; Number of entries in Multicast list
```

```
025D  1003 $DEF     CDB_B_MLTTBL     .BLKB   1          ; Number of entries in MLTTBL
025E  1004 $DEF     CDB_G_MULTI      .BLKW   3*MAX_C_MLT ; Multicast address table
02A6  1005 $DEF     CDB_G_MLTTBL     .BLKW   3*MAX_C_MLT ; Multicast address generation table
02EE  1006 $DEF     CDB_B_AQUOTA     .BLKB   1          ; Additional QUOTA available
02EF  1007 $DEF     CDB_B_MQUOTA     .BLKB   1          ; Maximum extra QUOTA allowed
02F0  1008
02F0  1009 $DEF     CDB_C_LENGTH                        ; Size of QNA CDB
02F0  1010
02F0  1011          _VIELD  CDB_STS,0,<-                ; CDB status bit for CDB_B_STS
02F0  1012                  <INITED,,M>,-               ; Inited
02F0  1013                  <RUN,,M>,-                  ; Device is running
02F0  1014                  <FORK_PEND,,M>,-            ; Fork process is pending
02F0  1015                  <TIMER,,M>,-                ; Timer is active
02F0  1016                  <ERR,,M>,-                  ; Fatal error has occurred
02F0  1017                  <SETUP,,M>,-                ; Device is in SETUP mode
02F0  1018                  >
02F0  1019
02F0  1020          _VIELD  CDB_MOD,0,<-                ; Bits for mode definition CDB_W_MODE
02F0  1021                  <MULTI,,M>,-                ; All multicasts enabled
02F0  1022                  <PROM,,M>,-                 ; Promiscuous mode enabled
02F0  1023                  >
02F0  1024
02F0  1025          $DEFEND CDB                         ; End of QNA CDB definitions
```

```
                        1027 ;
                        1028 ; P2 buffer header definition
                        1029 ;
                        1030         $DEFINI P2B
                        1031
0000                    1032 $DEF    P2B_L_POINTER   .BLKL    1       ; Pointer to start of data
0004                    1033 $DEF    P2B_L_BUFFER    .BLKL    1       ; Address of user's data buffer
0008                    1034 $DEF    P2B_W_SIZE      .BLKW    1       ; Size of P2 buffer
000A                    1035 $DEF    P2B_B_TYPE      .BLKB    1       ; Type of structure
000B                    1036 $DEF    P2B_B_SPARE     .BLKB    1       ; Spare byte
000C                    1037 $DEF    P2B_C_LENGTH                     ; Size of P2 buffer header
000C                    1038 $DEF    P2B_T_DATA                       ; Start of data
000C                    1039
000C                    1040         $DEFEND P2B
                        1041 ;
                        1042 ; Diagnostics buffer definition
                        1043 ;
                        1044         $DEFINI DIAG
                        1045 ;
                        1046 ; Driver independent portion of diagnostics buffer
                        1047 ;
0000                    1048 $DEF    DIAG_L_DATA     .BLKL    1       ; Pointer to start of data
0004                    1049 $DEF    DIAG_L_BUFFER   .BLKL    1       ; User buffer address
0008                    1050 $DEF    DIAG_W_SIZE     .BLKW    1       ; Size of structure
000A                    1051 $DEF    DIAG_B_TYPE     .BLKB    1       ; Type of structure
000B                    1052 $DEF    DIAG_B_SPARE    .BLKB    1       ; Spare byte
000C                    1053 $DEF    DIAG_T_DATA                      ; Start of data
000C                    1054 $DEF    DIAG_Q_START    .BLKQ    1       ; Start time for QIO
0014                    1055 $DEF    DIAG_Q_FINISH   .BLKQ    1       ; Finish time for QIO
001C                    1056 $DEF    DIAG_L_ERRS     .BLKL    1       ; Number of device errors
0020                    1057 $DEF    DIAG_L_EXTRA    .BLKL    1       ; Number of longwords that follow
0024                    1058 ;
0024                    1059 ; Driver dependent portion of diagnostics buffer
0024                    1060 ;
0024                    1061 $DEF    DIAG_L_DEPEND
0024                    1062 $DEF    DIAG_W_CSR      .BLKW    1       ; Last port CSR contents
0026                    1063 $DEF    DIAG_W_ERR      .BLKW    1       ; Ring entry error summary
0028                    1064 $DEF    DIAG_W_ERR2     .BLKW    1       ; Extra ring entry error info
002A                    1065 $DEF    DIAG_G_HWA      .BLKW    3       ; Hardware physical address
0030                    1066 ;
0030                    1067 ; The following is valid only on read (receive) QIOs
0030                    1068 ;
0030                    1069 $DEF    DIAG_T_RDATA                     ; Start of receive data
0030                    1070 $DEF    DIAG_G_DEST     .BLKW    3       ; Destination address
0036                    1071 $DEF    DIAG_G_SRC      .BLKW    3       ; Source address
003C                    1072 $DEF    DIAG_W_TYPE     .BLKW    1       ; Protocol type
003E                    1073 $DEF    DIAG_C_LENGTH                    ; Start of data
00000006                003E 1074 DIAG_C_EXTRA = .-DIAG_L_DEPEND/4
                        003E 1075
                        003E 1076         $DEFEND DIAG
                        1077 ;
                        1078 ; Receive buffer header definition
                        1079 ;
                        1080         $DEFINI RHDR
                        1081
0000                    1082 $DEF    RHDR_L_DATA     .BLKL    1       ; Pointer to start of data
0004                    1083 $DEF    RHDR_L_BUFFER   .BLKL    1       ; User buffer address
```

```
                     0008  1084 $DEF     RHDR_W_SIZE     .BLKW   1       ; Size of structure
                     000A  1085 $DEF     RHDR_B_TYPE     .BLKB   1       ; Type of structure
                     000B  1086 $DEF     RHDR_B_SPARE    .BLKB   1       ; Spare byte
                     000C  1087 $DEF     RHDR_T_DATA                     ; Start of data
                     000C  1088 $DEF     RHDR_G_DEST     .BLKW   3       ; Destination address
                     0012  1089 $DEF     RHDR_G_SRC      .BLKW   3       ; Source address
                     0018  1090 $DEF     RHDR_W_TYPE     .BLKW   1       ; Protocol type
                     001A  1091 $DEF     RHDR_C_LENGTH                   ; Start of data
0000000E             001A  1092 RHDR_C_DATA = .-RHDR_T_DATA
                     001A  1093
                     001A  1094         $DEFEND RHDR
                     0000  1095
                     0000  1096 ;
                     0000  1097 ; Shareable protocol type data structure
                     0000  1098 ;
                     0000  1099         $DEFINI SHR     GLOBAL
                     0000  1100 $DEF     SHR_L_QFL       .BLKL   1       ; Forward link pointer
                     0004  1101 $DEF     SHR_L_QBL       .BLKL   1       ; Backward link pointer
                     0008  1102 $DEF     SHR_W_SIZE      .BLKW   1       ; Size of structure
                     000A  1103 $DEF     SHR_B_TYPE      .BLKB   1       ; Type of structure
                     000B  1104 $DEF     SHR_B_STS       .BLKB   1       ; SHR status
                     000C  1105 $DEF     SHR_L_PID       .BLKL   1       ; User's PID
                     0010  1106 $DEF     SHR_W_CHAN      .BLKW   1       ; User's channel
                     0012  1107 $DEF     SHR_G_DEST      .BLKW   3       ; Destination address
                     0018  1108 $DEF     SHR_Q_QUEUES                    ; Start of queues
                     0018  1109 $DEF     SHR_Q_RCVMSG    .BLKQ   1       ; Received messages waiting for IRPs
                     0020  1110 $DEF     SHR_Q_RCVREQ    .BLKQ   1       ; Receive IRPs waiting for messages
00000002             0028  1111 SHR_C_QUEUES = <.-SHR_Q_QUEUES>/8         ; Number of queues
                     0028  1112 $DEF     SHR_W_QUOTA     .BLKW   1       ; User's shared quota
                     002A  1113 $DEF     SHR_C_LENGTH                    ; Size of data structure
                     002A  1114
                     002A  1115         _VIELD  SHR_STS,0,<-            ; SHR status bits
                     002A  1116                 <INITED,,M>,-           ; Protocol type is inited
                     002A  1117                 >
                     002A  1118
                     002A  1119         $DEFEND SHR
```

```
                        1121          .SBTTL  Standard tables
                        1122 ;
                        1123 ; Driver prologue table
                        1124 ;
0000                    1125          DPTAB   -                               ;
0000                    1126                  END=XQ_END,-                    ; END OF DRIVER
0000                    1127                  ADAPTER=UBA,-                   ; ADAPTER TYPE
0000                    1128                  UCBSIZE=UCB$C_XQ_LENGTH,-       ; SIZE OF UCB
0000                    1129                  NAME=XQDRIVER                   ; DRIVER NAME
0038                    1130
0038                    1131          DPT_STORE INIT                          ; START OF CONTROLLER INIT
0038                    1132          DPT_STORE UCB,UCB$B_FIPL,B,IPL$_XQ_FIPL ; FORK IPL
003C                    1133          DPT_STORE UCB,UCB$B_DIPL,B,IPL$_XQ_DIPL ; DEVICE IPL
0040                    1134          DPT_STORE ORB,ORB$B_FLAGS,B,-           ; Protection block flags
0040                    1135                    <ORB$M_PROT_16>               ; SOGW protection word
0044                    1136          DPT_STORE ORB,ORB$W_PROT,W,0            ; default protection
0049                    1137          DPT_STORE ORB,ORB$L_OWNER,L,0           ; no owner as yet
0050                    1138          DPT_STORE UCB,UCB$L_DEVCHAR,L,- ; DEVICE CHARACTERISTICS
0050                    1139                    <DEV$M_SHR!-
0050                    1140                     DEV$M_NET!-
0050                    1141                     DEV$M_AVL!-
0050                    1142                     DEV$M_IDV!-
0050                    1143                     DEV$M_ODV>
0057                    1144
0057                    1145          DPT_STORE UCB,UCB$B_DEVCLASS,B,DC$_SCOM ; Device class
005B                    1146          DPT_STORE UCB,UCB$B_DEVTYPE,B,DT$_DEQNA ; Device type
005F                    1147          DPT_STORE UCB,UCB$W_DEVBUFSIZ,W,512     ; Default buffer size
0064                    1148          DPT_STORE UCB,UCB$W_STS,W,<UCB$M_ONLINE!UCB$M_TEMPLATE>
0069                    1149          DPT_STORE UCB,UCB$G_XQ_PHA,L,-1         ; No default physical address
0070                    1150          DPT_STORE UCB,UCB$G_XQ_PHA+4,W,-1       ;       ...
0075                    1151 ;
0075                    1152 ; Store defaults for all parameters
0075                    1153 ;
;RNG0001                 0075    .1            DPT_STORE UCB,UCB$W_XQ_BSZ,W,MAX_PKT_SIZE ; Default device buffer size
-1                      007A    1155          DPT_STORE UCB,UCB$B_XQ_BFN,B,1          ; Default user buffer number
                        007E    1156          DPT_STORE UCB,UCB$W_XQ_HBQ,W,INIT_C_QUOTA ; Hardware Buffer Quota
                        0083    1157          DPT_STORE UCB,UCB$B_XQ_PRO,B,NMA$C_LINPR_NI ; "NI" is the protocol mode
                        0087    1158          DPT_STORE UCB,UCB$B_XQ_PRM,B,NMA$C_STATE_OFF ; Promiscuous mode is OFF
                        008B    1159          DPT_STORE UCB,UCB$B_XQ_MLT,B,NMA$C_STATE_OFF ; All multicasts is OFF
                        008F    1160          DPT_STORE UCB,UCB$B_XQ_DCH,B,NMA$C_STATE_ON ; Data chaining is ON
                        0093    1161          DPT_STORE UCB,UCB$B_XQ_PAD,B,NMA$C_STATE_ON ; Padding is ON
                        0097    1162          DPT_STORE UCB,UCB$B_XQ_CON,B,NMA$C_LINCN_NOR ; Controler mode is NORMAL
                        009B    1163          DPT_STORE UCB,UCB$B_XQ_ACC,B,NMA$C_ACC_EXC ; Exclusive mode is default
                        009F    1164
                        009F    1165          DPT_STORE REINIT
                        009F    1166
                        009F    1167          DPT_STORE DDB,DDB$L_DDT,D,XQ$DDT        ; DDT ADDRESS
                        00A4    1168          DPT_STORE CRB,CRB$L_INTD+4,D,QNA_INTR   ; QNA inter ,pt service routine
                        00A9    1169          DPT_STORE CRB,CRB$L_INTD+VEC$L_INITIAL,D,CONTROL_INIT; CONTROLLER INIT ADDRE
                        00AE    1170          DPT_STORE CRB,CRB$L_INTD+VEC$L_UNITINIT,D,UNIT_INIT; UNIT INIT
                        00B3    1171          DPT_STORE CRB,CRB$L_INTD+VEC$L_START,D,FFI_INIT ; FFI INIT
                        00B8    1172          DPT_STORE END
                        0000    1173
                  00000000      1174          .PSECT  $$$115_DRIVER,LONG
                        0000    1175 ;
                        0000    1176 ; LOCAL STORAGE
                        0000    1177 ;
```

```
                        ; Driver dispatch table
                        ;
                DDTAB   DEVNAM=XQ,-                     ; DRIVER DISPATCH TABLE
                        START=STARTIO,-                 ; Start I/O operation
                        FUNCTB=XQ_FUNCTABLE,-           ; Function decision table address
                        CANCEL=CANCEL,-                 ; CANCEL I/O entry point
                        REGDMP=REG_DUMP,-               ; Register dump entry point
                        DIAGBF=<DIAG_C_LENGTH>,-        ; Diagnostic buffer size
                        CLONEDUCB=CLONED_UCB,-          ; Cloned UCB initialization
                        ALTSTART=ALT_START              ; Alternate start I/O entry point
                ;
                ; Function decision table
                ;
XQ_FUNCTABLE:                                           ;
                FUNCTAB ,-                              ; Legal functions
                <WRITEVBLK,WRITELBLK,WRITEPBLK,READVBLK,READLBLK,-
                READPBLK,SENSEMODE,SENSECHAR,SETMODE,SETCHAR>
                FUNCTAB ,-                              ; Buffered functions
                <WRITEVBLK,WRITELBLK,WRITEPBLK,READVBLK,READLBLK,-
                READPBLK,SENSEMODE,SENSECHAR,SETMODE,SETCHAR>
                FUNCTAB XMT_FDT,<WRITELBLK,WRITEPBLK,WRITEVBLK> ;
                FUNCTAB RCV_FDT,<READLBLK,READPBLK,READVBLK> ;
                FUNCTAB SETMODE_FDT,<SETMODE,SETCHAR> ;
                FUNCTAB SENSEMODE_FDT,<SENSEMODE,SENSECHAR> ;
```

```
                     0078  1204          .SBTTL  Local driver storage
                     0078  1205 ;
                     0078  1206 ; P2 Buffer verification tables
                     0078  1207 ;
                     0078  1208
                     0078  1209          $DEFINI PARAM
                     0000  1210
                     0000  1211 $DEF     PRM_W_TYPE      .BLKW   1       ; Parameter type
                     0002  1212
                     0002  1213          _VIELD  PRM_TYP,0,<-            ; Parameter type field
                     0002  1214                  <CODE,12,M>,-           ; Parameter type code
                     0002  1215                  <STRING,1,M>,-          ; Parameter is a string
                     0002  1216                  >
                     0002  1217
                     0002  1218 $DEF     PRM_B_FLAG      .BLKB   1       ; Parameter flags
                     0003  1219
                     0003  1220          _VIELD  PRM_FLG,0,<-            ; Parameter flag bits
                     0003  1221                  <MIN,1,M>,-             ; Parameter minimum value present
                     0003  1222                  <MAX,1,M>,-             ; Parameter maximum value present
                     0003  1223                  <INVALID,1,M>,-         ; Parameter invalid value is present
                     0003  1224                  <CDB,1,M>,-             ; Offset is in CDB data base
                     0003  1225                  <CHECK,1,M>,-           ; Check values with current
                     0003  1226                  >
                     0003  1227
                     0003  1228 $DEF     PRM_W_OFF       .BLKW   1       ; Parameter offset in structure
                     0005  1229
                     0005  1230          _VIELD  PRM_OFF,0,<-            ; Offset word fields
                     0005  1231                  <VALUE,10,M>,-          ; Offset value
                     0005  1232                  <WIDTH,6,M>,-           ; Size of field in structure
                     0005  1233                  >
                     0005  1234
                     0005  1235          $DEFEND PARAM
                     0078  1236
                     0078  1237 ;
                     0078  1238 ; Define Line parameters
                     0078  1239 ;
00000000             0078  1240 LINE_PRM_BUFSIZ=0                       ; Line parameter buffer size
                     0078  1241 LINE_PARAM_WO:                          ; "Write-Only" line parameters
                     0078  1242
                     0078  1243          PARAM   NMA$C_PCLI_HBQ,-        ; Hardware Buffer Quota
                     0078  1244                  OFFSET=UCB$W_XQ_HBQ,-
                     0078  1245                  WIDTH=W,MAX=16384,-
                     0078  1246                  INVALID=UCB$M_XQ_INITED
                     0081  1247
                     0081  1248 LINE_PARAM:                             ; Start of line parameters
                     0081  1249
                     0081  1250          PARAM   NMA$C_PCLI_ACC,-        ; Access mode for protocol type
                     0081  1251                  OFFSET=UCB$B_XQ_ACC,-
                     0081  1252                  WIDTH=B,-
                     0081  1253                  MIN=NMA$C_ACC_SHR,-
                     0081  1254                  MAX=NMA$C_ACC_EXC
                     008A  1255
                     008A  1256          PARAM   NMA$C_PCLI_PRO,-        ; Protocol selection mode
                     008A  1257                  OFFSET=UCB$B_XQ_PRO,-
                     008A  1258                  WIDTH=B,-
                     008A  1259                  MIN=NMA$C_LINPR_POI,-   ;% Accept either point or NI
                     008A  1260                  MAX=NMA$C_LINPR_NI
```

```
                        1261
                        1262    PARAM   NMA$C_PCLI_BUS,-        ; Buffer size
                        1263            OFFSET=UCB$W_DEVBUFSIZ,-
                        1264            WIDTH=W,-
                        1265            MIN=MIN_PKT_SIZE,-      ; User buffer LIMITS
                        1266            MAX=MAX_PKT_SIZE,-
                        1267            INVALID=UCB$M_XQ_INITED
                        1268
                        1269    PARAM   NMA$C_PCLI_BFN,-        ; Buffer number
                        1270            OFFSET=UCB$B_XQ_BFN,-
                        1271            WIDTH=B,-
                        1272            MIN=0,MAX=255,-
                        1273            INVALID=UCB$M_XQ_INITED
                        1274
                        1275    PARAM   NMA$C_PCLI_PHA,-        ; Physical NI address
                        1276            OFFSET=UCB$G_XQ_PHA,-
                        1277            STRING=YES,-
                        1278            SIZE=<2+6>,-
                        1279            INVALID=UCB$M_XQ_INITED
                        1280
                        1281    PARAM   NMA$C_PCLI_DCH,-        ; Data chaining on receives
                        1282            OFFSET=UCB$B_XQ_DCH,-
                        1283            WIDTH=B,-
                        1284            MAX=NMA$C_STATE_OFF
                        1285
                        1286    PARAM   NMA$C_PCLI_PAD,-        ; Padding mode
                        1287            OFFSET=UCB$B_XQ_PAD,-
                        1288            WIDTH=B,-
                        1289            MAX=NMA$C_STATE_OFF
                        1290
                        1291    PARAM   NMA$C_PCLI_PRM,-        ; Promiscuous mode state
                        1292            OFFSET=UCB$B_XQ_PRM,-
                        1293            WIDTH=B,-
                        1294            MAX=NMA$C_STATE_OFF,-
                        1295            INVALID=UCB$M_XQ_INITED
                        1296
                        1297    PARAM   NMA$C_PCLI_MLT,-        ; Accept all multicast addresses
                        1298            OFFSET=UCB$B_XQ_MLT,-
                        1299            WIDTH=B,-
                        1300            MAX=NMA$C_STATE_OFF,-
                        1301            INVALID=UCB$M_XQ_INITED
                        1302
                        1303    PARAM   NMA$C_PCLI_CON,-        ; Controller mode
                        1304            OFFSET=UCB$B_XQ_CON,-
                        1305            WIDTH=B,-
                        1306            MAX=NMA$C_LINCN_LOO,-
                        1307            INVALID=UCB$M_XQ_INITED
                        1308
                        1309    PARAM   NMA$C_PCLI_PTY,-        ; Protocol type
                        1310            OFFSET=UCB$W_XQ_PROTYP,-
                        1311            WIDTH=W,-
                        1312            INVALID=UCB$M_XQ_INITED
                        1313
                        1314    PARAM   NMA$C_PCLI_MCA,-        ; Multicast address list
                        1315            OFFSET=UCB$G_XQ_MULTI,-
                        1316            STRING=YES,-
                        1317            SIZE=2+<6*MAX_C_MLT>    ; Maximum size of list
```

```
00E5  1318
00E5  1319          PARAM   NMA$C_PCLI_BSZ,-          ; Device buffer size
00E5  1320                  OFFSET=UCB$W_XQ_BSZ,-
00E5  1321                  WIDTH=W,-
00E5  1322                  MIN=MIN_PKT_SIZE,-
00E5  1323                  MAX=MAX_PKT_SIZE,-
00E5  1324                  INVALID=UCB$M_XQ_INITED
00F0  1325
00F0  1326          PARAM   NMA$C_PCLI_DES,-          ; Destination Address for shared
00F0  1327                  OFFSET=UCB$G_XQ_DES,-     ;  Protocol Type
00F0  1328                  STRING=YES,-
00F0  1329                  SIZE=<2+6>
00F5  1330
00F5  1331 ;****
00F5  1332 ; THE FOLLOWING CAN BE ELIMINATED
00F5  1333 ;****
00F5  1334          PARAM   NMA$C_PCLI_CRC,-          ; CRC enabled
00F5  1335                  OFFSET=UCB$B_XQ_MST,-     ;  garbage
00F5  1336                  WIDTH=B,-                 ;
00F5  1337                  MAX=NMA$C_STATE_OFF
00FC  1338
00FC  1339          PARAM                             ; End of table
00FE  1340
00FE  1341 CIRCUIT_PARAM:                             ; Start of circuit parameter table
00FE  1342
00FE  1343          PARAM   NMA$C_PCCI_MST,-          ; Maintenance state
00FE  1344                  OFFSET=UCB$B_XQ_MST,-
00FF  1345                  WIDTH=B,-
00FE  1346                  MIN=NMA$C_STATE_ON,-
00FE  1347                  MAX=NMA$C_STATE_OFF
0107  1348
0107  1349          PARAM                             ; End of table
0109  1350
0109  1351 ;
0109  1352 ; Line/circuit counters
0109  1353 ;
0109  1354 LINE_CTR:                                  ; Start of LINE counters
0109  1355          COUNTER         ZER, 16, ZERO     ; Seconds since last zeroed
010D  1356          COUNTER         DBR, 32, DBRCTR   ; Packets received
0111  1357          COUNTER         MBL, 32, MBLCTR   ; Multicast packets received
0115  1358          COUNTER         RFL, 16, RFLMAP,, MAP ; Packets received in error
0119  1359          COUNTER         BRC, 32, BRCCTR   ; Bytes received
011D  1360          COUNTER         MBY, 32, MBYCTR   ; Multicast bytes received
0121  1361          COUNTER         OVR, 16, OVRCTR   ; Receives lost - Internal buffer error
0125  1362          COUNTER         LBE, 16, LBECTR   , Receives lost - Local buffer error
0129  1363          COUNTER         DBS, 32, DBSCTR   ; Packets transmitted
012D  1364          COUNTER         MBS, 32, MBSCTR   ; Multicast packets transmitted
0131  1365          COUNTER         BSM, 32, BSMCTR   ; Packets transmitted - several errors
0135  1366          COUNTER         BS1, 32, BS1CTR   ; Packets transmitted - 1 error
0139  1367          COUNTER         BID, 32, BIDCTR   ; Packets transmitted - deferred
013D  1368          COUNTER         BSN, 32, BSNCTR   ; Bytes transmitted
0141  1369          COUNTER         MSN, 32, MSNCTR   ; Multicast bytes transmitted
0145  1370          COUNTER         SFL, 16, SFLMAP,, MAP ; Transmit packets aborted
0149  1371          COUNTER         CDC, 16, CDCCTR   ; Transmit collision check failure
014D  1372          COUNTER         UFD, 16, UFDCTR   ; Unrecognized frame destination
0151  1373          COUNTER         SBU, 16, SBUCTR   ; System buffer unavailable
0155  1374          COUNTER         UBU, 16, UBUCTR   ; User buffer unavailable
```

```
                     0159  1375
                     0159  1376 CIRC_CTR:                                ; Start of CIRCUIT counters
                     0159  1377          COUNTER         DBS, 32, SBLCTR, CIRC ; Blocks sent
                     015D  1378          COUNTER         BSN, 32, SBYCTR, CIRC ; Bytes sent
                     0161  1379          COUNTER         DBR, 32, RBLCTR, CIRC ; Blocks received
                     0165  1380          COUNTER         BRC, 32, RBYCTR, CIRC ; Bytes received
                     0169  1381          COUNTER         MNE, 16, MNECTR, CIRC ; Multicast address not enabled
                     016D  1382          COUNTER         UBU, 16, UBUCTR, CIRC ; User buffer unavailable
                     0171  1383
                     0171  1384 ;
                     0171  1385 ; MOP read counters return table (in order of COUNTERs returned)
                     0171  1386 ;
                     0171  1387 MOPCTRTAB:                                ; Start of MOP counters
                     0171  1388          MOPCTR          16, ZERO        ; Seconds since last zeroed
                     0174  1389          MOPCTR          32, BRCCTR      ; Bytes received
                     0177  1390          MOPCTR          32, BSNCTR      ; Bytes transmitted
                     017A  1391          MOPCTR          32, DBRCTR      ; Packets received
                     017D  1392          MOPCTR          32, DBSCTR      ; Packets transmitted
                     0180  1393          MOPCTR          32, MBYCTR      ; Multicast bytes received
                     0183  1394          MOPCTR          32, MBLCTR      ; Multicast packets received
                     0186  1395          MOPCTR          32, BIDCTR      ; Packets transmitted - deferred
                     0189  1396          MOPCTR          32, BS1CTR      ; Packets transmitted - 1 error
                     018C  1397          MOPCTR          32, BSMCTR      ; Packets transmitted - several errors
                     018F  1398          MOPCTR          16, SFLCTR, MAP ; Transmit packets aborted
                     0192  1399          MOPCTR          16, RFLCTR, MAP ; Packets received in error
                     0195  1400          MOPCTR          16, UFDCTR      ; Unrecognized frame destination
                     0198  1401          MOPCTR          16, OVRCTR      ; Receives lost - Internal buffer error
                     019B  1402          MOPCTR          16, LBECTR      ; Receives lost - Local buffer error
                     019E  1403          MOPCTR          16, UBUCTR      ; User buffer unavailable
                     01A1  1404          MOPCTR          16, CDCCTR      ; Transmit collision check failure
                     01A4  1405          MOPCTR                          ; End of table
                     01A6  1406 ;
                     01A6  1407 ; Calculate total size of MOP counter return data buffer
                     01A6  1408 ;
           00000043  01A6  1409          MOP_CTR_SIZE = MOP_CTR_SIZE + 1 + 2 + 8 ; Size of counters + MOP header
           00000051  01A6  1410          MOP_CTR_SIZE = MOP_CTR_SIZE + XBUF_C_HEADER ; Size of buffer + NI header
                     01A6  1411
                     01A6  1412 ;
                     01A6  1413 ; BAD PARAMETER RETURN TABLE
                     01A6  1414 ;
                     01A6  1415 ; First part is validation of Unit against controller. The second part is
                     01A6  1416 ; for validation of shared protocol types.
                     01A6  1417 ;
                     01A6  1418 ; Note that the table is in the REVERSE order from that of the UCB.
                     01A6  1419 ;
                     01A6  1420          ASSUME  UCB$B_XQ_CON EQ UCB$B_XQ_CDBPRM
                     01A6  1421 BAD_PARAM_TBL:
                0456 01A6  1422          .WORD   NMA$C_PCLI_CON
                     01A8  1423
                     01A8  1424          ASSUME  UCB$W_XQ_BSZ EQ UCB$B_XQ_SHRPRM
                     01A8  1425          ASSUME  UCB$B_XQ_PRO EQ UCB$W_XQ_BSZ+2
                     01A8  1426          ASSUME  UCB$B_XQ_PAD EQ UCB$B_XQ_PRO+1
                     01A8  1427          ASSUME  UCB$B_XQ_PRM EQ UCB$B_XQ_PAD+1
                     01A8  1428          ASSUME  UCB$B_XQ_MLT EQ UCB$B_XQ_PRM+1
                     01A8  1429          ASSUME  UCB$B_XQ_DCH EQ UCB$B_XQ_MLT+1
                     01A8  1430          ASSUME  UCB$B_XQ_CDBPRM EQ UCB$B_XQ_DCH+1
                0818 01A8  1431          .WORD   NMA$C_PCLI_DCH
```

```
0B19  01AA  1432          .WORD   NMA$C_PCLI_MLT
0B18  01AC  1433          .WORD   NMA$C_PCLI_PRM
0B1A  01AE  1434          .WORD   NMA$C_PCLI_PAD
0458  01B0  1435          .WORD   NMA$C_PCLI_PRO
0B20  01B2  1436          .WORD   NMA$C_PCLI_BSZ
0B20  01B4  1437          .WORD   NMA$C_PCLI_BSZ          ; This parameter is a word (not byte)
```

```
                01B6  1439          .SBTTL  CONTROL_INIT - INITIALIZE DEQNA DEVICE
                01B6  1440 ;++
                01B6  1441 ; CONTROL_INIT - INITIALIZE DEQNA DEVICE
                01B6  1442 ;
                01B6  1443 ; Functional description:
                01B6  1444 ;
                01B6  1445 ; This routine is entered when driver is loaded, system is booted, or
                01B6  1446 ; during powerfail recovery.
                01B6  1447 ;
                01B6  1448 ; Inputs:
                01B6  1449 ;
                01B6  1450 ;       R4 = Address of the device CSR
                01B6  1451 ;       R5 = Address of the device IDB
                01B6  1452 ;       R6 = Address of the device DDB
                01B6  1453 ;       R8 = Address of the device CRB
                01B6  1454 ;
                01B6  1455 ;       IPL = FIPL
                01B6  1456 ;
                01B6  1457 ; Outputs:
                01B6  1458 ;
                01B6  1459 ;       R4,R5,R8 are preserved
                01B6  1460 ;--
                01B6  1461
                01B6  1462 CONTROL_INIT::                          ; Initialize the DEQNA
     05         01B6  1463          RSB                           ; Return to caller
```

```
                     01B7  1465          .SBTTL  CLONED_UCB - INITIALIZE THE CLONED UCB
                     01B7  1466 ;++
                     01B7  1467 ; CLONED_UCB - INITIALIZE THE CLONED UCB
                     01B7  1468 ;
                     01B7  1469 ; Functional description:
                     01B7  1470 ;
                     01B7  1471 ; This routine is called by the $ASSIGN system service to allow the driver
                     01B7  1472 ; to initialize the cloned UCB. The driver is called with process context.
                     01B7  1473 ;
                     01B7  1474 ; Inputs:
                     01B7  1475 ;
                     01B7  1476 ;       R0 = SS$_NORMAL
                     01B7  1477 ;       R2 = UCB address of cloned UCB
                     01B7  1478 ;       R3 = DDT address
                     01B7  1479 ;       R4 = PCB address
                     01B7  1480 ;       R5 = UCB address of template UCB
                     01B7  1481 ;
                     01B7  1482 ;       IPL = ASTDEL
                     01B7  1483 ;
                     01B7  1484 ; Outputs:
                     01B7  1485 ;
                     01B7  1486 ;       R0 = SS$_NORMAL
                     01B7  1487 ;       R5 = UCB address of cloned UCB
                     01B7  1488 ;       All other registers and IPL are preserved.
                     01B7  1489 ;--
                     01B7  1490
                     01B7  1491 CLONED_UCB::                                 ; Cloned UCB initialization
     55    52    D0  01B7  1492          MOVL    R2,R5                      ; Copy UCB address
                     01BA  1493                                             ; Continue in unit_initialization
```

```
                        01BA  1495          .SBTTL  UNIT_INIT - INITIALIZE THE DEQNA UNIT
                        01BA  1496 ;++
                        01BA  1497 ; UNIT_INIT - INITIALIZE THE DEQNA UNIT
                        01BA  1498 ;
                        01BA  1499 ; Functional description:
                        01BA  1500 ;
                        01BA  1501 ; This routine is called at system startup, during driver loading and
                        01BA  1502 ; during powerfail recovery to initialize the DEQNA unit and its UCB.
                        01BA  1503 ; The UCB is initialized and if power has failed, the device is forced
                        01BA  1504 ; to shutdown.
                        01BA  1505 ;
                        01BA  1506 ; Inputs:
                        01BA  1507 ;
                        01BA  1508 ;       R4 = CSR address
                        01BA  1509 ;       R5 = UCB address
                        01BA  1510 ;
                        01BA  1511 ;       IPL = FIPL
                        01BA  1512 ;
                        01BA  1513 ; Outputs:
                        01BA  1514 ;
                        01BA  1515 ;       None.
                        01BA  1516 ;--
                        01BA  1517
                        01BA  1518 UNIT_INIT::                          ; Initialize a DEQNA unit
                        01BA  1519 ;;&&     JSB     G^INI$BRK           ;*** TEMP ***
                        01BA  1520
              3F   BB   01BA  1521          PUSHR   #^M<R0,R1,R2,R3,R4,R5>  ; Save all regs
    19 64 A5  05   E0   01BC  1522          BBS     #UCB$V_POWER,UCB$W_STS(R5),15$ ; Br if powerfail
                        01C1  1523
                        01C1  1524 ;
                        01C1  1525 ; Initialize UCB queue listheads, and the pointer (within the NI device
                        01C1  1526 ; dependent UCB extension, UCB$L_NI_MLTPTR) to the multicast address table for
                        01C1  1527 ; this protocol type.
                        01C1  1528 ;
                        01C1  1529
        00EC C5   9E    01C1  1530          MOVAB   UCB$G_XQ_MULTI(R5),-    ; Initialize the pointer to this
        0094 C5         01C5  1531                  UCB$L_NI_MLTPTR(R5)     ; protocol's multicast address table
                        01C8  1532
        50   04   D0    01C8  1533          MOVL    #UCB$C_XQ_QUEUES,R0     ; Get number of queue listheads in UCB
  51    0098 C5   9E    01CB  1534          MOVAB   UCB$Q_XQ_QUEUES(R5),R1  ; Get address of queue listheads
        81   61   DE    01D0  1535 10$:     MOVAL   (R1),(R1)+              ; Set forward link pointer
    81    FC A1   D0    01D3  1536          MOVL    -4(R1),(R1)+            ; Set backward link pointer
          F6 50   F5    01D7  1537          SOBGTR  R0,10$                  ; Loop if more listheads
                        01DA  1538
          54 A5   B5    01DA  1539 15$:     TSTW    UCB$W_UNIT(R5)          ; Is this unit 0?
             15   13    01DD  1540          BEQL    17$                     ; Br if yes - leave TEMPLATE bit on
        0082 C5   B4    01DF  1541          CLRW    UCB$W_ERRCNT(R5)        ; Only unit #0 may list errors
                        01E3  1542 ;
                        01E3  1543 ; We must find the address of unit 0, so we can check if the QNA is ONLINE.
                        01E3  1544 ; If the QNA is OFFLINE, then we mark each UCB as being OFFLINE.
                        01E3  1545 ;
  50    28 A5   D0    01E3  1546          MOVL    UCB$L_DDB(R5),R0        ; Get address of QNA DDB
  50    04 A0   D0    01E7  1547          MOVL    DDB$L_UCB(R0),R0        ; Get address of UNIT 0 UCB
             04   E0    01EB  1548          BBS     #UCB$V_ONLINE,-         ; Br if QNA is ONLINE
    04 64 A0          01ED  1549                  UCB$W_STS(R0),17$       ;
             10   AA    01F0  1550          BICW    #UCB$M_ONLINE,-         ; Else, mark new unit as OFFLINE
          64 A5         01F2  1551                  UCB$W_STS(R5)           ;
```

```
                                01F4  1552 17$:     ;
;RNG0001                        01F4    .1          ; For MicroVAX I, we will have to allocate a physically contiguous
;RNG0001                        01F4    .2          ; buffer area for perfoming I/O on the QNA.
;RNG0001                        01F4    .3          ;
;RNG0001                        01F4    .4          CPUDISP <<790,60$>,-
;RNG0001                        01F4    .5                  <780,60$>,-
;RNG0001                        01F4    .6                  <750,60$>,-
;RNG0001                        01F4    .7                  <730,60$>,-
;RNG0001                        01F4    .8                  <UV2,60$>,-
;RNG0001                        01F4    .9                  <UV1,20$>>                  ; For MicroVAX I, allocate buffer area
;RNG0001                        0210    .10                                             ; For all others, skip buffer area
;RNG0001                        0210    .11
;RNG0001                        0210    .12 20$:    ;  MicroVAX I
;RNG0001                        0210    .13
;RNG0001         54   00C8 C5   D0  0210    .14         MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
;RNG0001                 0B     12  0215    .15         BNEQ    23$                     ; Br if present
;RNG0001                1C99    30  0217    .16         BSBW    ALLOC_CDB               ; Else, try to allocate a CDB
;RNG0001              21 50     E9  021A    .17         BLBC    R0,60$                  ; Br if error
;RNG0001         54   00C8 C5   D0  021D    .18         MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-16                  020C C4    D5  0222 1569 23$:      TSTL    CDB_L_UV1BUF(R4)        ; Is the buffer area allocated?
                           16   12  0226 1570           BNEQ    60$                     ; Br if yes, continue
                                0228 1571
                      00002400  0228 1572 UV1_BUFFER_LENGTH = <UV1_BUFFER_AREA + 511> & <^C511>    ; Round to a page
                      00000012  0228 1573 UV1_BUFFER_PAGES  = UV1_BUFFER_LENGTH / 512             ; Number of pages
                                0228 1574
                      51   12   3C  0228 1575           MOVZWL  #UV1_BUFFER_PAGES,R1    ; Number of pages to allocate
                00000000'GF    16  022B 1576           JSB     G^EXE$ALOPHYCNTG        ; Allocate physically-contiguous memory
                        0A 50  E9  0231 1577           BLBC    R0,60$                  ; Skip ahead on error
;RNG0001         54   00C8 C5   D0  0234    .1          MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2               020C C4   52    D0  0239 1580           MOVL    R2,CDB_L_UV1BUF(R4)     ; Save buffer area address
                                023E 1581
                      50   24   9A  023E 1582 60$:      MOVZBL  #3*MAX_C_MLT,R0         ; Get size of multicast list in words
                51    00EC C5   9E  0241 1583           MOVAB   UCB$G_XQ_MULTI(R5),R1   ; Get address of multicast list
                           81   B4  0246 1584 70$:      CLRW    (R1)+                   ; Init multicast table
                         FB 50  F5  0248 1585           SOBGTR  R0,70$                  ; Loop if more
;RNG0001         54   00C8 C5   D0  024B    .1          MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2                        06   12  0250 1588           BNEQ    80$                     ; Br if present
                        1C5E   30  0252 1589           BSBW    ALLOC_CDB               ; Else, try to allocate a CDB
                      0F 50    E9  0255 1590           BLBC    R0,90$                  ; Br if no error
                0A 64 A5       05  E1  0258 1591 80$:      BBC     #UCB$V_POWER,UCB$W_STS(R5),90$ ; Br if not powerfail
                  00024000 8F  D0  025D 1592           MOVL    #<XQ_SOFT_M_POWERAT6>!- ; Indicate cause of error
                           53      0263
                                   0264 1593                   XQ_CSR_M_ERR,R3
                        13DF   30  0264 1594           BSBW    SCHED_FORK              ; Schedule fork process
                          3F   BA  0267 1595 90$:      POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore regs
                               05  0269 1596           RSB                             ; Done
```

```
                     026A  1598          .SBTTL  FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
                     026A  1599 ;++
                     026A  1600 ; FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
                     026A  1601 ;
                     026A  1602 ; Functional description:
                     026A  1603 ;
                     026A  1604 ; This routine initializes the FFI interface. Currently the UCB must
                     026A  1605 ; have been initialized prior to calling this routine, in the future
                     026A  1606 ; this routine may have to initialize the UCB and DEQNA. Therefore,
                     026A  1607 ; there may be a fork involved in the call to this routine.
                     026A  1608 ;
                     026A  1609 ; Inputs:
                     026A  1610 ;
                     026A  1611 ;       R3 = Address of quadword descriptor for parameter buffer
                     026A  1612 ;       R4 = FFI block address
                     026A  1613 ;
                     026A  1614 ;       IPL = SYNCH
                     026A  1615 ;
                     026A  1616 ; Outputs:
                     026A  1617 ;
                     026A  1618 ;       R0 = Status of request
                     026A  1619 ;       All other registers are preserved.
                     026A  1620 ;
                     026A  1621 ;--
                     026A  1622          ASSUME  IPL$_SYNCH EQ IPL$_XQ_FIPL
                     026A  1623 FFI_INIT::
            3E   BB  026A  1624          PUSHR   #^M<R1,R2,R3,R4,R5>     ; Save registers
            50   D4  026C  1625          CLRL    R0                      ; Assume failure
     55  34   A4   D0  026E  1626          MOVL    FFI$L_DL_UCB(R4),R5     ; Get UCB address
            04   E1  0272  1627          BBC     #UCB$V_XQ_RUN,-         ; Br if device not ready
                                                UCB$W_DEVSTS(R5),90$
      0E 68  A5       0274  1628
       0346'CF   9E   0277  1629          MOVAB   W^XMT_FFI_START,-       ; Return address of XMIT routine
          10  A4       027B  1630                  FFI$L_XMIT(R4)
0192  C5    54   D0   027D  1631          MOVL    R4,UCB$L_XQ_FFI(R5)     ; Save FFI address
      50     01   9A   0282  1632          MOVZBL  #1,R0                   ; Return success
            3E   BA   0285  1633 90$:     POPR    #^M<R1,R2,R3,R4,R5>     ; Restore registers
                 05   0287  1634          RSB                             ; Return to caller
                      0288  1635
```

```
                     0288  1637          .SBTTL  XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
                     0288  1638 ;++
                     0288  1639 ; XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
                     0288  1640 ;
                     0288  1641 ; Functional description:
                     0288  1642 ;
                     0288  1643 ; This routine sets up the internal function code for transmit and
                     0288  1644 ; transfers control to the exec buffered I/O write FDT routine.
                     0288  1645 ;
                     0288  1646 ; The QIO parameters for WRITES are:
                     0288  1647 ;
                     0288  1648 ;       P1 = Address of the data buffer
                     0288  1649 ;       P2 = Size of the data buffer
                     0288  1650 ;       P5 = Address of buffer containing the destination address
                     0288  1651 ;
                     0288  1652 ;  ** The driver can never do direct I/O on XMIT requests, because  **
                     0288  1653 ; *** the QNA buffer address cannot begin on an odd byte boundary.  ***
                     0288  1654 ;  **    Also, the FAST interface cannot operate on DIRECT I/O.     **
                     0288  1655 ;
                     0288  1656 ;
                     0288  1657 ; Inputs:
                     0288  1658 ;
                     0288  1659 ;       R3 = IRP address
                     0288  1660 ;       R4 = PCB address
                     0288  1661 ;       R5 = UCB address
                     0288  1662 ;       R6 = CCB address
                     0288  1663 ;       R7 = FUNCTION CODE
                     0288  1664 ;
                     0288  1665 ;       IPL = ASTDEL
                     0288  1666 ;
                     0288  1667 ; Outputs:
                     0288  1668 ;
                     0288  1669 ;       R0-R2,R8,R9 are destroyed.
                     0288  1670 ;
                     0288  1671 ;--
                     0288  1672 ABORTIO_BR:                                 ; Long branch to ABORTIO
          00B5   31  0288  1673          BRW     ABORTIO                    ; Abort the I/O request
                     0289  1674
                     028B  1675 XMT_FDT::                                   ; Transmit FDT routine
        40 A3   7C   028B  1676          CLRQ    IRP$Q_STATION(R3)          ; Zero the destination address
      00CE C5   B0   028E  1677          MOVW    UCB$W_XQ_PROTYP(R5),-      ; Assume we are a non-promiscuous user
         3A A3        0292  1678                  IRP$W_XQ_PROTYP(R3)        ;
    51   10 AC   D0   0294  1679          MOVL    P5(AP),R1                  ; Get address of destination address
         05   12      0298  1680          BNEQ    10$                        ; Br if given
                     029A  1681 ;
                     029A  1682 ; If the user is in shared mode, then he does not have to supply a destination
                     029A  1683 ; address with each transmit operation. The destination address will be gotten
                     029A  1684 ; from the SHR data structure.
                     029A  1685 ;
         03   E0      029A  1686          BBS     #UCB$V_XQ_SHARE,-          ; Br if shared user
      12 68 A5        029C  1687                  UCB$W_DEVSTS(R5),20$       ;
       50   0C   9A   029F  1688 10$:     MOVZBL  S^#SS$_ACCVIO,R0           ; Assume access violation
                     02A2  1689          IFNORD  #6,(R1),ABORTIO_BR         ; Check access to buffer
    40 A3   61   D0   02A8  1690          MOVL    (R1),IRP$Q_STATION(R3)     ; Save destination address
44 A3   04 A1   B0   02AC  1691          MOVW    4(R1),IRP$Q_STATION+4(R3)  ;     ...
                     02B1  1692
                     02B1  1693          ASSUME  NMA$C_STATE_ON EQ 0
```

```
                     02B1  1694          ASSUME  NMA$C_STATE_OFF EQ 1
                     02B1  1695
 0D 00DE C5    E8    02B1  1696 20$:     BLBS    UCB$B_XQ_PRM(R5),30$     ; Br if user is not promiscuous
    51    06    C0    02B5  1697          ADDL    #6,R1                    ; Point to protocol type
                     02B8  1698          IFNORD  #2,(R1),ABORTIO_BR       ; Check access to buffer
 3A A3    61    B0    02BF  1699          MOVW    (R1),IRP$W_XQ_PROTYP(R3) ; Get protocol type from user P5 buffer
    50    14    9A    02C3  1700 30$:     MOVZBL  S^#SS$_BADPARAM,R0       ; Assume bad parameters
    58    6C    D0    02C6  1701          MOVL    P1(AP),R8                ; Get starting address of user buffer
 59   04 AC    3C    02C9  1702          MOVZWL  P2(AP),R9                ; Get length of user buffer
          71    13    02CD  1703          BEQL    ABORTIO                  ; Br if zero length buffer
    50    58    7D    02CF  1704          MOVQ    R8,R0                    ; Retrieve buffer parameters
00000000'GF    16    02D2  1705          JSB     G^EXE$WRITECHK           ; Check accessibility of user buffer
                     02D8  1706                                           ; (No return on NO ACCESS)
                     02D8  1707                                           ; Returns IRP$W_BCNT
    2E    51    B1    02D8  1708          CMPW    R1,#MIN_PKT_SIZE         ; Is buffer at least minimum?
          03    1E    02DB  1709          BGEQU   50$                      ; Br if yes, okay
    51    2E    3C    02DD  1710          MOVZWL  #MIN_PKT_SIZE,R1         ; Else, allocate minimum sized packet
00000048 8F    C0    02E0  1711 50$:     ADDL2   #CXB$C_HEADER,R1         ; Calculate length of buffer needed
          51          02E6
          38    BB    02E7  1712          PUSHR   #^M<R3,R4,R5>            ; Save registers
00000000'GF    16    02E9  1713          JSB     G^EXE$BUFFRQUOTA         ; Check if process has sufficient quota
       4C 50    E9    02EF  1714          BLBC    R0,90$                   ; Br if quota check failure
00000000'GF    16    02F2  1715          JSB     G^EXE$ALLOCBUF           ; Allocate CXB buffer for output
       43 50    E9    02F8  1716          BLBC    R0,90$                   ; If LBC allocation failure
    53    6E    D0    02FB  1717          MOVL    (SP),R3                  ; Retrieve address of IRP
50   0080 C4    D0    02FE  1718          MOVL    PCB$L_JIB(R4),R0         ; Get JIB address
  20 A0    51    C2    0303  1719          SUBL    R1,JIB$L_BYTCNT(R0)      ; Adjust buffered I/O quota
  30 A3    51    B0    0307  1720          MOVW    R1,IRP$W_BOFF(R3)        ; Set number of bytes charged to quota
  2C A3    52    D0    030B  1721          MOVL    R2,IRP$L_SVAPTE(R3)      ; Save CXB address in IRP
          52    DD    030F  1722          PUSHL   R2                       ; Save pointer to CXB
                     0311  1723
                     0311  1724          ASSUME  CXB$L_FL EQ 0
                     0311  1725          ASSUME  CXB$L_BL EQ CXB$L_FL+4
          82    7C    0311  1726          CLRQ    (R2)+                    ; Clear link cells
                     0313  1727
                     0313  1728          ASSUME  CXB$W_SIZE EQ CXB$L_BL+4
    82    51    B0    0313  1729          MOVW    R1,(R2)+                 ; Set size of structure
                     0316  1730
                     0316  1731          ASSUME  CXB$B_TYPE EQ CXB$W_SIZE+2
                     0316  1732          ASSUME  CXB$B_CODE EQ CXB$B_TYPE+1
    82    1B    9B    0316  1733          MOVZBW  #DYN$C_CXB,(R2)+         ; Set structure type
                     0319  1734
    52    6E    D0    0319  1735          MOVL    (SP),R2                  ; Get back CXB address
                     031C  1736
                     031C  1737          ASSUME  CXB$C_HEADER EQ CXB$T_T_DATA+XQ_C_HEADER
  18 A2    3A    B0    031C  1738          MOVW    #CXB$T_T_DATA,CXB$W_BOFF(R2) ; Setup offset to start of data
  52   48 A2    9E    0320  1739          MOVAB   CXB$C_HEADER(R2),R2      ; Get address of data portion of buffer
                     0324  1740
62    68    59    28    0324  1741          MOVC3   R9,(R8),(R2)             ; Move data to system buffer
          3C    BA    0328  1742          POPR    #^M<R2,R3,R4,R5>         ; Restore registers
                     032A  1743          SETIPL  UCB$B_FIPL(R5)           ; Sync access to UCB
          53    DD    032E  1744          PUSHL   R3                       ; Save IRP address
          57    10    0330  1745          BSBB    XMT_START                ; Do common processing
          53 8ED0     0332  1746          POPL    R3                       ; Restore IRP address
       08 50    E9    0335  1747          BLBC    R0,ABORTIO               ; Br if error in processing request
00000000'GF    17    0338  1748          JMP     G^EXE$QIORETURN          ; Exit QIO service to await completion
                     033E  1749
```

```
                 38   BA  033E  1750 90$:     POPR    #^M<R3,R4,R5>           ; Restore registers
00000000'GF      17   0340  1751 ABORTIO:JMP     G^EXE$ABORTIO           ; Abort the I/O request
```

```
                     0346  1753          .SBTTL  XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
                     0346  1754 ;++
                     0346  1755 ; XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
                     0346  1756 ;
                     0346  1757 ; Functional description:
                     0346  1758 ;
                     0346  1759 ; This routine is called to start a transmit operation. If the QNA is running
                     0346  1760 ; then the request is given to the xmit wait queue for the QNA. If there is
                     0346  1761 ; a free entry in the transmit ring and there are sufficient map registers to
                     0346  1762 ; map the buffer then the request is given to the QNA immediately, else the
                     0346  1763 ; request is left on the xmit wait queue until another request completes.
                     0346  1764 ;
                     0346  1765 ;
                     0346  1766 ; Inputs:
                     0346  1767 ;
                     0346  1768 ;       R3 = CXB address
                     0346  1769 ;       R4 = FFI address
                     0346  1770 ;
                     0346  1771 ;       IPL = SYNCH (same as FIPL)
                     0346  1772 ;
                     0346  1773 ; Outputs:
                     0346  1774 ;
                     0346  1775 ;       R0,R3 are destroyed.
                     0346  1776 ;       All other registers are preserved.
                     0346  1777 ;
                     0346  1778 ;       If the request cannot be queued, then the FFI$L_XMIT_DONE entry is
                     0346  1779 ;       called immediately with the following:
                     0346  1780 ;
                     0346  1781 ;               R0 = Status of transmit request
                     0346  1782 ;               R3 = CXB address
                     0346  1783 ;               R4 = FFI address
                     0346  1784 ;
                     0346  1785 ;--
                     0346  1786          ASSUME  IPL$_SYNCH EQ IPL$_XQ_FIPL
                     0346  1787 XMT_FFI_START::                          ; Start FAST interface transmit request
          36   BB    0346  1788          PUSHR   #^M<R1,R2,R4,R5>        ; Save registers
    55    34 A4   D0  0348  1789          MOVL    FFI$L_DL_UCB(R4),R5     ; Get UCB address
                     034C  1790          ;
                     034C  1791          ;  For the FFI Interface, we will save the UCB address
                     034C  1792          ;
    24 A3   55   D0  034C  1793          MOVL    R5,CXB$L_T_UCB(R3)      ; Save UCB address
                     0350  1794          ASSUME  CXB$L_T_UCB  EQ  CXB$L_T_IRP
    24 A3   01   88  0350  1795          BISB    #1,CXB$L_T_UCB(R3)      ; ...indicate this is a UCB address
                     0354  1796
    18 A3   0E   A2  0354  1797          SUBW    #XQ_C_HEADER,CXB$W_BOFF(R3) ; Back up offset for Ethernet header
                     0358  1798
                     0358  1799          ASSUME  NMA$C_STATE_ON EQ 0
                     0358  1800          ASSUME  NMA$C_STATE_OFF EQ 1
    04 00DD C5   E8  0358  1801          BLBS    UCB$B_XQ_PAD(R5),20$    ; Br if padding is disabled
    18 A3   02   A2  035D  1802          SUBW    #XQ_C_CNTSIZ,CXB$W_BOFF(R3) ; Else, skip length field of buffer
 51    18 A3   3C  0361  1803 20$:       MOVZWL  CXB$W_BOFF(R3),R1       ; Get offset to start of data
52  53   51   C1  0365  1804          ADDL3   R1,R3,R2                ; Set R2 to start of header
                     0369  1805
                     0369  1806          ASSUME  XBUF_G_SRC EQ XBUF_G_DEST+6
        28 A3   7D  0369  1807          MOVC    CXB$Q_STATION(R3),-     ; Store destination address
           62           036C  1808                  XBUF_G_DEST(R2)
                     036D  1809          ASSUME  XBUF_Q_TYPE EQ XBUF_G_SRC+6
```

```
                00E2 C5   7D   036D  1810          MOVQ    UCB$G_XQ_PHA(R5),-      ; Store our source address
                  06 A2         0371  1811            XBUF_G_SRC(R2)
                00CE C5   B0   0373  1812          MOVW    UCB$W_XQ_PROTYP(R5),-   ; Set PROTOCOL TYPE
                  0C A2         0377  1813            XBUF_W_TYPE(R2)
                      53   10   0379  1814          BSBB    XMT_INITIATE            ; Try to start transmit
                   08 50   E8   037B  1815          BLBS    R0,50$                  ; Br if success
          54    0192 C5   D0   037E  1816          MOVL    UCB$L_XQ_FFI(R5),R4     ; Else, get back FFI address
                  14 B4   16   0383  1817          JSB     @FFI$C_XMIT_DONE(R4)    ; Complete request in error
                     36   BA   0386  1818 50$:     POPR    #^M<R1,R2,R4,R5>        ; Restore registers
                          05   0388  1819          RSB                             ; Return to caller
```

```
                                0389  1821          .SBTTL  XMT_START - START TRANSMIT OPERATION
                                0389  1822 ;++
                                0389  1823 ; XMT_START - START TRANSMIT OPERATION
                                0389  1824 ;
                                0389  1825 ; Functional description:
                                0389  1826 ;
                                0389  1827 ; This routine is called to start a transmit operation. If the QNA is running
                                0389  1828 ; then the request is given to the xmit wait queue for the QNA. If there is
                                0389  1829 ; a free entry in the transmit ring and there are sufficient map registers to
                                0389  1830 ; map the buffer then the request is given to the QNA immediately, else the
                                0389  1831 ; request is left on the xmit wait queue until another request completes.
                                0389  1832 ;
                                0389  1833 ;
                                0389  1834 ;  ** The driver can never do direct I/O on XMIT requests, because  **
                                0389  1835 ; *** the QNA buffer address cannot begin on an odd byte boundary.  ***
                                0389  1836 ;  **    Also, the FAST interface cannot operate on DIRECT I/O.     **
                                0389  1837 ;
                                0389  1838 ;
                                0389  1839 ; Inputs:
                                0389  1840 ;
                                0389  1841 ;       R2 = CXB address
                                0389  1842 ;       R3 = IRP address
                                0389  1843 ;       R5 = UCB address
                                0389  1844 ;
                                0389  1845 ;       IPL = FIPL
                                0389  1846 ;
                                0389  1847 ; Outputs:
                                0389  1848 ;
                                0389  1849 ;       R0 = Status of transmit request
                                0389  1850 ;       R1,R2,R4 are destroyed.
                                0389  1851 ;
                                0389  1852 ;--
                                0389  1853          .ENABL  LSB
                                0389  1854 XMT_START::                                ; Start transmit operation
                         04     0389  1855          BBS     #UCB$V_XQ_RUN,-         ; Br if unit is in RUN mode
                   10 68 A5     038B  1856                  UCB$W_DEVSTS(R5),30$
                                038E  1857
                  50   20D4 8F 3C   038E  1858 10$:    MOVZWL  #SS$_DEVINACT,R0        ; Assume unit not started yet
                         0E     E1   0393  1859          BBC     #UCB$V_XQ_INTERLOCK,-   ; Br if unit is not re-starting
                   05 68 A5     0395  1860                  UCB$W_DEVSTS(R5),20$    ;  on it's own.
                  50   02D4 8F 3C   0398  1861          MOVZWL  #SS$_OPINCOMPL,R0       ; Else, return different error code
                         05     0D   039D  1862 20$:    RSB                             ; Okay to leave now
                                039E  1863
                     24 A2  53   D0   039E  1864 30$:    MOVL    R3,CXB$L_T_IRP(R2)      ; Save IRP address
                      0094 C3   D4   03A2  1865          CLRL    IRP$L_XQ_SETUP(R3)      ; Indicate no SETUP buffer present
                1A A2   32 A3   B0   03A6  1866          MOVW    IRP$W_BCNT(R3),CXB$W_BCNT(R2) ; Set BCNT in CXB
;RNG0001                 53   52   7D   03AB     .1      MOVQ    R2,R3                   ; Copy IRP, CXB addresses
-2
                                03AE  1869
                                03AE  1870          ASSUME  NMA$C_STATE_ON EQ 0
                                03AE  1871          ASSUME  NMA$C_STATE_OFF EQ 1
                     04 00DD C5   E8   03AE  1872          BLBS    UCB$B_XQ_PAD(R5),40$    ; Br if padding is disabled
                     18 A3   02   A2   03B3  1873          SUBW    #XQ_C_CNTSIZ,CXB$W_BOFF(R3) ; Else, skip length field of buffer
                     51   18 A3   3C   03B7  1874 40$:    MOVZWL  CXB$W_BOFF(R3),R1       ; Get offset to start of data
                  52   53   51   C1   03BB  1875          ADDL3   R1,R3,R2                ; Set R2 to start of header
                                03BF  1876          ASSUME  XBUF_G_SRC EQ XBUF_G_DEST+6
                        40 A4   7D   03BF  1877          MOVQ    IRP$Q_STATION(R4),-     ; Store destination address
                           62        03C2  1878                   XBUF_G_DEST(R2)
```

```
                     03C3  1879          ASSUME  XBUF_W_TYPE EQ XBUF_G_SRC+6
00E2 C5   7D         03C3  1880          MOVQ    UCB$G_XQ_PHA(R5),-      ; Store our source address
  06 A2              03C7  1881                   XBUF_G_SRC(R2)
  3A A4   B0         03C9  1882          MOVW    IRP$W_XQ_PROTYP(R4),-   ; Store PROTOCOL TYPE
  0C A2              03CC  1883                   XBUF_W_TYPE(R2)
                     03CE  1884
                     03CE  1885 XMT_INITIATE::                           ; FAST Interface entry point (FFI)
                     03CE  1886          ;
                     03CE  1887          ; Inputs:
                     03CE  1888          ;       R5 = UCB address
                     03CE  1889          ;       R3 = CXB address
                     03CE  1890          ;       R2 = Start address for Ethernet header
                     03CE  1891          ;
20 A3   01   90      03CE  1892          MOVB    #XQ_FC_V_XMIT,CXB$B_XQ_FUNC(R3) ; Set function request in CXB
                     03D2  1893
                     03D2  1894          ASSUME  NMA$C_STATE_ON EQ 0
                     03D2  1895          ASSUME  NMA$C_STATE_OFF EQ 1
09 00DD C5   E8      03D2  1896          BLBS    UCB$B_XQ_PAD(R5),60$    ; Br if padding is disabled
                     03D7  1897 ;
                     03D7  1898 ;  PADDING IS ENABLED:
                     03D7  1899 ;       Adjust byte count to include size field and store count field.
                     03D7  1900 ;
  1A A3   B0         03D7  1901          MOVW    CXB$W_BCNT(R3),-        ; Else, store size of data-only
  0E A2              03DA  1902                  XBUF_W_SIZE(R2)         ;  portion of buffer in message
1A A3   02   A0      03DC  1903          ADDW    #XQ_C_CNTSIZ,CXB$W_BCNT(R3) ; And account for count field
                     03E0  1904 ;
                     03E0  1905 ; Allow buffer size up to Ethernet max buffer size for transmit operations.
                     03E0  1906 ;
  1A A3   B1         03E0  1907 60$:     CMPW    CXB$W_BCNT(R3),#MAX_PKT_SIZE ; Is buffer size bigger than
05DC 8F              03E3
                     03E6  1908                                          ;  largest Ethernet buffer allowed?
        06   1B      03E6  1909          BLEQU   90$                     ; Br if no
50   034C 8F   3C    03E8  1910 80$:     MOVZWL  #SS$_IVBUFLEN,R0        ; Assume bad buffer length
                     03ED  1911          RSB                             ; ELSE, leave now
                     03EE  1912
;RNG0001         54  00C8 C5   D0    03EE    .1 90$:     MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2                       01   E1    03F3  1915          BBC     #CDB_STS_V_RUN,-        ; Br if QNA not running
                   1B 0246 C4       03F5  1916                  CDB_B_STS(R4),100$      ;
                        0E   A0     03F9  1917          ADDW    #XQ_C_HEADER,-          ; Adjust byte count
                      1A A3         03FB  1918                   CXB$W_BCNT(R3)         ;  for header info
                                    03FD  1919 ;
                                    03FD  1920 ; If running in the SHARED-LIMITED mode, then we must use the destination
                                    03FD  1921 ; address from the SHR_ data structure. Unless the given destination address
                                    03FD  1922 ; is a multicast address. For the SHARED-DEFAULT user, we must make sure that
                                    03FD  1923 ; destination address given is unique!
                                    03FD  1924 ;
                        03   F1     03FD  1925          BBC     #UCB$V_XQ_SHARE,-       ; Br if NOT a shared user
                   4E 68 A5         03FF  1926                   UCB$W_DEVSTS(R5),NO_SHR ;
                                    0402  1927 ;
                                    0402  1928 ; Try to find a match on PID/CHAN. Returns pointer in R1
                                    0402  1929 ;
                                    0402  1930          ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
                   0E 24 A3   E8    0402  1931          BLBS    CXB$L_T_IRP(R3),100$    ; Br if FFI user, return failure
                         53   DD    0406  1932          PUSHL   R3                      ; Else, save CXB address
                   53    24 A3   D0  0408  1933          MOVL    CXB$L_T_IRP(R3),R3      ; Get IRP address
                        0301   30    040C  1934          BSBW    MATCH_SHR               ; Try to find the SHR data structure
                          06   13    040F  1935          BEQL    110$                    ; Br if match
```

```
                53 8ED0  0411  1936          POPL    R3                              ; Restore CXB address
            FF77   31    0414  1937 100$:    BRW     10$                             ; Else, error
                       0417  1938
                53 8ED0  0417  1939 110$:    POPL    R3                              ; Restore CXB address
    51  00C4   C5   D1  041A  1940          CMPL    UCB$L_XQ_DEFUSR(R5),R1          ; Is this the default user?
            0E   13      041F  1941          BEQL    SHR_DEF                         ; Br if yes
                       0421  1942 ;
                       0421  1943 ; This is a SHARED-LIMITED user.
                       0421  1944 ;
            2C 62    E8  0421  1945          BLBS    XBUF_G_DEST(R2),NO_SHR          ; Br if multicast address
            12 A1    D0  0424  1946          MOVL    SHR_G_DEST(R1),-                ; Else, get destination from SHR struct.
               62        0427  1947                    XBUF_G_DEST(R2)               ;       ...
            16 A1    B0  0428  1948          MOVW    SHR_G_DEST+4(R1),-              ;       ...
            04 A2        042B  1949                    XBUF_G_DEST+4(R2)             ;       ...
               21    11  042D  1950          BRB     NO_SHR                          ; Continue in common code
                       042F  1951          .DSABL  LSB
                       042F  1952 ;
                       042F  1953 ; This is a SHARED-DEFAULT user.
                       042F  1954 ;
    50  0098   C5   9E  042F  1955 SHR_DEF:MOVAB    UCB$Q_XQ_SHARE(R5),R0           ; Get address of SHR listhead
        51   50  D0  0434  1956          MOVL    R0,R1                           ; Save address of start of listhead
        51   61  D0  0437  1957 10$:     MOVL    SHR_L_QFL(R1),R1                ; Get address of next in list
        51   50  D1  043A  1958          CMPL    R0,R1                           ; Back at start of list?
             11  13  043D  1959          BEQL    NO_SHR                          ; Br if yes, destination is unique
             62  D1  043F  1960          CMPL    XBUF_G_DEST(R2),-               ; Address match?
          12 A1      0441  1961                  SHR_G_DEST(R1)                  ;       ...
             F2  12  0443  1962          BNEQ    10$                             ; Br if no - check next in list
          04 A2  B1  0445  1963          CMPW    XBUF_G_DEST+4(R2),-             ; Hi order of address match?
          16 A1      0448  1964                  SHR_G_DEST+4(R1)                ;       ...
             EB  12  044A  1965          BNEQ    10$                             ; Br if not - check next in list
        50   14  3C  044C  1966          MOVZWL  #SS$_BADPARAM,R0                ; Else, bad parameter code
                 05  044F  1967          RSB                                     ; Return to caller
                       0450  1968
                       0450  1969 NO_SHR:
                       0450  1970          ;
                       0450  1971          ; Do accounting for MULTICAST here. NOTE: this should really
                       0450  1972          ; be done upon completion of request, but it is done here to
                       0450  1973          ; save extra work to check for multicast in the completion
                       0450  1974          ; section.
                       0450  1975          ;
          1A 62    E9  0450  1976          BLBC    XBUF_G_DEST(R2),20$             ; Br if NOT multicast address
                       0453  1977          INCC    CDB_L_MBSCTR(R4)                ; Count multicast blocks sent
    50    1A A3    3C  045D  1978          MOVZWL  CXB$W_BCNT(R3),R0               ; Get BCNT
                       0461  1979          CNTR    R0,CDB_L_MSNCTR(R4),L           ; Count multicast bytes sent
                       046D  1980 20$:
                       046D  1981 ;
                       046D  1982 ; If we are running in point-to-point mode, then queue xmit on wait queue
                       046D  1983 ; if we are waiting for run!
                       046D  1984 ;
              00   91  046D  1985          CMPB    #NMA$C_LINPR_POI,-              ;% Are we in PT-TO-PT mode?
        00DC  C5       046F  1986                  UCB$B_XQ_PRO(R5)                ;%
              0E   12  0472  1987          BNEQ    40$                             ;% Br if not
              06   E1  0474  1988          BBC     #UCB$V_XQ_STACK,-               ;% Br if not in stack wait state
     09 68    A5       0476  1989                  UCB$W_DEVSTS(R5),40$            ;%
00B4 D5       63   0E  0479  1990          INSQUE  (R3),@UCB$Q_XQ_XMTREQ+4(R5);% Else, insert request on wait queue
        50    01   9A  047E  1991          MOVZBL  #1,R0                           ;% Return success
                 05  0481  1992          RSB                                     ;% Return to caller
```

```
                                0482  1993 ;
                                0482  1994 ; Insert request on CDB transmit request queue and check if transmit
                                0482  1995 ; can proceed.
                                0482  1996 ;
                     60E0 D4    63   0E  0482  1997 40$:    INSQUE  (R3),@CDB_Q_XMTREQ+4(R4) ; Insert at end of xmit queue
                                0487  1998
                                0487  1999         .ENABL  LSB
                                0487  2000 XMT_ALT_START::                          ; Alternate start for xmit
:RNG0001                        0487     .1         ;
:RNG0001                        0487     .2         ; Inputs:
:RNG0001                        0487     .3         ;       R5 = UCB address
:RNG0001                        0487     .4         ;       R4 = CDB address
:RNG0001                        0487     .5         ;
:RNG0001                        0487     .6         PUSHQ   R6                      ; Save R6,R7
:RNG0001                 56  24 A5   D0  048A     .7         MOVL    UCB$L_CRB(R5),R6        ; Get CRB address
:RNG0001                        048E     .8 ;
:RNG0001                        048E     .9 ; Skip MAP register useage if MicroVAX I. Also use different number of slots.
:RNG0001                        048E    .10 ;
:RNG0001                        048E    .11         CPUDISP <<790,10$>,-
:RNG0001                        048E    .12                 <780,10$>,-
:RNG0001                        048E    .13                 <750,10$>,-
:RNG0001                        048E    .14                 <730,10$>,-
:RNG0001                        048E    .15                 <UV2,10$>,-
:RNG0001                        048E    .16                 <UV1,XMT_UV1>>
:RNG0001                        04AA    .17
:RNG0001                        04AA    .18 10$:    ;  All but MicroVAX I
:RNG0001                        04AA    .19
:RNG0001                        04AA    .20         ASSUME  MAX_C_XMT LE 8
-13                        03   00   EB  04AA  2014         FFC     #0,#MAX_C_XMT-1,CDB_B_XMTMAP(R4),R7 ; Find a free transmit slot
                        57  1B A4        04AD
                               4C   13  04B0  2015         BEQL    20$                     ; Br if none free
                                        04B2  2016 ;
                                        04B2  2017 ; Move CXB info into UCB
                                        04B2  2018 ;
                         53   00DC D4   0F  04B2  2019         REMQUE  @CDB_Q_XMTREQ(R4),R3    ; Get oldest xmit request
                                     45   1D  04B7  2020         BVS     20$                     ; Br if none
:RNG0001                          1A A3   B1  04B9     .1         CMPW    CXB$W_BCNT(R3),-        ; Is packet at least minimum size?
:RNG0001                             3C       04BC     .2                 #MIN_PKT_SIZE+XQ_C_HEADER
:RNG0001                             04   1E  04BD     .3         BGEQU   15$                     ; Br if yes, okay
:RNG0001                             3C   B0  04BF     .4         MOVW    #MIN_PKT_SIZE+XQ_C_HEADER,- ; Else, set to minimum
:RNG0001                          1A A3       04C1     .5                 CXB$W_BCNT(R3)
:RNG0001                 7E A5    1A A3   B0  04C3     .6 15$:    MOVW    CXB$W_BCNT(R3),UCB$W_BCNT(R5) ; Set byte count
-1                          52    18 A3   3C  04C8  2022         MOVZWL  CXB$W_BOFF(R3),R2       ; Get offset to start of data
                               52    53   C0  04CC  2023         ADDL    R3,R2                   ; Compute buffer virtual address
                               FE00  8F   AB  04CF  2024         BICW3   #^C<VA$M_BYTE>,-        ; Get buffer offset
                            7C A5    52       04D3  2025                 R2,UCB$W_BOFF(R5)       ;
                                        04D6  2026 ;
                                        04D6  2027 ; Convert virtual address to physical PTE address
                                        04D6  2028 ;
                                     09   EF  04D6  2029         EXTZV   S^#VA$V_VPN,-           ; Get virtual page number
                                52   52   15  04D8  2030                 S^#VA$S_VPN,R2,R2       ;
                                 00000000'GF  D0  04DB  2031         MOVL    G^MMG$GL_SPTBASE,R0     ; Get the base address of the SPT
                                        50       04E1
                                78 A5   6042  DE  04E2  2032         MOVAL   (R0)[R2],UCB$L_SVAPTE(R5) ; Set address of the SPT entry
                                        04E7  2033 ;
                                        04E7  2034 ; The following instruction also sets the data path number to the Direct
                                        04E7  2035 ; Data Path.
```

```
                                04E7  2036 ;
                                04E7  2037          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                04E7  2038          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                     38 A4   D0 04E7  2039          MOVL    CDB_L_XMTMAP(R4),-      ; Assume we use preallocated map
                     34 A6      04EA  2040                  CRB$L_INTD+VEC$W_MAPREG(R6) ; register.
                        57   D5 04EC  2041          TSTL    R7                      ; Is mapping slot the preallocated one?
                        1B   13 04EE  2042          BEQL    50$                     ; Br if yes - all set
                                04F0  2043                                          ; Else, allocate the map registers
                                04F0  2044 ;
                                04F0  2045 ; Allocate UNIBUS map registers
                                04F0  2046 ;
                                04F0  2047 ;;&&     CLRB    CRB$L_INTD+VEC$B_DATAPATH(R6) ; Reset data path usage
                00000000'GF   16 04F0  2048          JSB     G^IOC$ALOUBAMAP         ; Allocate UNIBUS map registers
                         0C   50 E8 04F6  2049          BLBS    R0,40$                  ; Br if one available
                00DC C4      63 0E 04F9  2050          INSQUE  (R3),CDB_Q_XMTREQ(R4)   ; Re-insert CXB on request queue
                                04FE  2051 20$:     POPQ    R6                      ; Restore R6,R7
                     50   01   9A 0501  2052          MOVZBL  S^#SS$_NORMAL,R0        ; Good return
                                05 0504  2053 30$:     RSB                             ; Return to caller
                                0505  2054 ;
                                0505  2055 ; Save the map information and map the buffer.
                                0505  2056 ;
                                0505  2057 40$:     ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                0505  2058          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                        34 A6   D0 0505  2059          MOVL    CRB$L_INTD+VEC$W_MAPREG(R6),- ; Save mapping info
                     38 A447       0508  2060                  CDB_L_XMTMAP(R4)[R7]    ; in CDB
                                0508  2061 50$:     SETBIT  R7,CDB_B_XMTMAP(R4)     ; Set mapping slot in use flag
                  22 A3   57   90 0510  2062          MOVB    R7,CXB$B_XQ_SLOT(R3)    ; Save mapping slot number used
                00000000'GF   16 0514  2063          JSB     G^IOC$LOADUBAMAPA       ; Load map registers
                                051A  2064 ;
                                051A  2065 ; Find next ring entry and insert data
                                051A  2066 ;
                     52   18 A4   9A 051A  2067          MOVZBL  CDB_B_NEXTXMT(R4),R2    ; Get next ring entry
                          18 A4   96 051E  2068          INCB    CDB_B_NEXTXMT(R4)       ; Bump ring pointer
                          FC 8F   8A 0521  2069          BICB    #^C<MAX_C_XMT-1>,-      ; Modulo xmit ring size
                          18 A4      0524  2070                  CDB_B_NEXTXMT(R4)       ;
                     23 A3    52   90 0526  2071          MOVB    R2,CXB$B_XQ_RING(R3)    ; Save ring entry number
                52    009C C442   D0 052A  2072          MOVL    CDB_L_XRINGVA(R4)[R2],R2; Get ring entry virtual address
                             62   B4 0530  2073          CLRW    XMT_W_FLAG(R2)          ; Zero the FLAG word
                        8000 8F   B0 0532  2074          MOVW    #XMT_STS_M_LAST,-       ; Init STATUS word
                          08 A2      0536  2075                  XMT_Q_STS(R2)           ;
                     50   1A A3   3C 0538  2076          MOVZWL  CXB$W_BCNT(R3),R0       ; Get BYTE count
;RNG0001                   50   D6 053C     .1          INCL    R0                      ; Round up by one
-4                   50   FF 8F   78 053E  2081          ASHL    #-1,R0,R0               ; Convert to WORD count
                             50      0542
                     06 A2   50   AE 0543  2082          MNEGW   R0,XMT_W_LEN(R2)        ; Store message length (2's complement)
                  04 A2   7C   A5 B0 0547  2083          MOVW    UCB$W_BOFF(R5),XMT_W_ADDR(R2) ; Move byte offset - BA0-BA8
                          34 A6   F0 054C  2084          INSV    CRB$L_INTD+VEC$W_MAPREG(R6),- ; Insert BA9-BA15
                        07   09      054F  2085                  #9,#7,XMT_W_ADDR(R2)    ;&
                          04 A2      0551
                        06   07   EF 0553  2086          EXTZV   #7,#6,CRB$L_INTD+VEC$W_MAPREG(R6),R0 ; Get BA16-BA21
                     50   34 A6      0556
                     02 A2   50   B0 0559  2087          MOVW    R0,XMT_W_ADDRHI(R2)     ; Insert BA16-BA21 & zero descriptor bits
                        56   52   D0 055D  2088          MOVL    R2,R6                   ; Save xmit ring entry address
                                0560  2089                  ;
;RNG0001                         0560     .1          ; Descriptor bit settings are calculated as follows.
;RNG0001                         0560     .2          ;
;RNG0001                         0560     .3          ;  If the beginning address is Even or ODD, then the END address is:
```

```
;RNG0001                                0560     .4          ;
;RNG0001                                0560     .5          ;       BEGINNING  &  LENTH, then END address is:
;RNG0001                                0560     .6          ;       ---------     -----      ---
;RNG0001                                0560     .7          ;          Even          Even       Even
;RNG0001                                0560     .8          ;          Even          Odd        Odd
;RNG0001                                0560     .9          ;          Odd           Even       Odd
;RNG0001                                0560    .10          ;          Odd           Odd        Even
-8
                                        0560  2098          ;
                50    A000 8F    B0     0560  2099          MOVW    #XMT_DSC_M_VALID!-      ; Build descriptor flag
                                        0565  2100                    XMT_DSC_M_EOM,R0
                    11 7C A5     E9     0565  2101          BLBC    UCB$Q_BOFF(R5),70$      ; Br if even byte boundary at BEGinning
                                        0569  2102          ;
                                        0569  2103          ; Beginning buffer address is Odd
                                        0569  2104          ;
                                        0569  2105          SETBIT  #XMT_DSC_V_BEGODD,R0    ; Else, beginning is Odd
                    11 1A A3     E8     056D  2106          BLBS    CXB$Q_BCNT(R3),80$      ; Br if odd LENgth, end is even
                                        0571  2107          SETBIT  #XMT_DSC_V_ENDODD,R0    ; Else, ending is Odd
                       06 A6     B7     0575  2108          DECW    XMT_Q_LEN(R6)           ; Increase length by 1 (complemented)
                          08     11     0578  2109          BRB     80$                     ; Continue
                                        057A  2110          ;
                                        057A  2111          ; Beginning buffer address is Even
                                        057A  2112          ;
                    04 1A A3     E9     057A  2113 70$:     BLBC    CXB$W_BCNT(R3),80$      ; Br if even LENgth
                                        057E  2114          SETBIT  #XMT_DSC_V_ENDODD,R0    ; Else, ending is Odd
                          01     91     0582  2115 80$:     CMPB    #XQ_FC_V_XMIT,-         ; Is this a normal XMIT?
                       20 A3           0584  2116                  CXB$B_XQ_FUNC(R3)       ;
;RNG0001                  04     13     0586    .1          BEQL    90$                     ; Br if yes
;RNG0001                                0588    .2          SETBIT  #XMT_DSC_V_SETUP,R0     ; Else, indicate SETUP function
;RNG0001                     0A A6     B4     058C    .3 90$:     CLRW    XMT_Q_TDR(R6)           ; Clear the TDR cell
;RNG0001                 0107 C4       96     058F    .4          INCB    CDB_B_XMTCNT(R4)        ; Tally one more xmit in progress
;RNG0001            020A C4     04     90     0593    .5          MOVB    #XMT_TIM,CDB_B_TIM_XMT(R4) ; Start xmit timer
;RNG0001              02 A6     50     A8     0598    .6          BISW    R0,XMT_W_ADDRHI(R6)     ; Set descriptor bits
;RNG0001                                0598    .7 ;
;RNG0001                                0598    .8 ; Poll the xmit ring to see if it went invalid
;RNG0001                                0598    .9 ;
;RNG0001                                0598   .10          POPQ    R6                      ; Restore R6,R7
;RNG0001          52   0114 C4     D0     059B   .11          MOVL    CDB_L_CSR(R4),R2        ; Get CSR address
;RNG0001                                05A0   .12
;RNG0001                                05A0   .17 ;
;RNG0001                                05A0   .18 ;   If the QNA has invalidated the TRANSMIT RING, then we must reset the
;RNG0001                                05A0   .19 ;   starting address of the ring list to point to the current entry.
;RNG0001                                05A0   .20 ;
;RNG0001            0E A2     10     B3     05A0   .25          BITW    #XQ_CSR_M_XMTINV,CSR(R2); Is transmit ring still valid?
;RNG0001                        16     13     05A4   .26          BEQL    120$                    ; Br if yes
;RNG00C1              51    23 A3     9A     05A6   .28          MOVZBL  CXB$B_XQ_RING(R3),R1    ; Else, get ring entry number
;RNG0001          51    68 A441     D0     05AA   .29          MOVL    CDB_L_XRINGPA(R4)[R1],R1; Get the buffer mapping value
;RNG0001            08 A2     51     B0     05AF   .30          MOVW    R1,XMTLIST(R2)          ; Set address of transmit list entry
;RNG0001              51    F0 8F     78     05B3   .31          ASHL    #-16,R1,R1              ; Shift down high order address bits
                                51     05B7
;RNG0001              0A A2     51     90     05B8   .32          MOVB    R1,XMTLST1(R2)          ;                 ...
;RNG0001          00E8 D4     63     0E     05BC   .33 120$:     INSQUE  (R3),@CDB_Q_XMTPND+4(R4) ; Insert CXB on WAITING queue
;RNG0001                                05C1   .34 ;;&&     BITW    #XQ_CSR_M_CAR,CSR(R2)   ; Is carrier present?
;RNG0001                                05C1   .35 ;;&&     BEQL    130$                    ; Br if not, okay - else
;RNG0001                                05C1   .36 ;;&&     INCC    CDB_L_BIDCTR(R4)        ; Count blocks sent - initially deferred
;RNG0001          020A C4     04     90     05C1   .37 130$:     MOVB    #XMT_TIM,CDB_B_TIM_XMT(R4) ; Start xmit timer
;RNG0001                                05C6   .39
;RNG0001                   FEBA     31     05C6   .40          BRW     XMT_ALT_START           ; Try for more requests
```

```
:RNG0001                                05CD    .41
:RNG0001                                05CD    .42 XMT_UV1:
:RNG0001                                05CD    .43         ; Transmit operation on MicroVAX I
:RNG0001                                05CD    .44
:RNG0001                                05CD    .45         ASSUME  MAX_C_XMTUV1 LE 8
:RNG0001                                05CD    .46         ASSUME  MAX_C_XMTUV1 LT MAX_C_XMT ; Must not use all rings
:RNG0001          01    00    EB      05CD    .47         FFC     #0,#MAX_C_XMTUV1,CDB_B_XMTMAP(R4),R7 ; Find a free transmit slot
                57    1B A4             05D0
:RNG0001                    7D    13    05D3    .48         BEQL    220$                    ; Br if none free
:RNG0001                                05D5    .49 ;
:RNG0001                                05D5    .50 ; Move CXB data to contiguous buffer.
:RNG0001                                05D5    .51 ;
:RNG0001          53    00DC D4     0F  05D5    .52         REMQUE  @CDB_Q_XMTREQ(R4),R3    ; Get oldest xmit request
:RNG0001                    76    1D    05DA    .53         BVS     220$                    ; Br if none
-26               52    18 A3       3C  05DC  2143         MOVZWL  CXB$W_BOFF(R3),R2       ; Get offset to start of data
                    52    53        C0  05E0  2144         ADDL    R3,R2                   ; Compute system buffer virtual address
                51    00D8 C447     D0  05E3  2145         MOVL    CDB_L_XMT_VA(R4)[R7],R1 ; Get contiguous buffer's VA
                            78      BB  05E9  2146         PUSHR   #^M<R3,R4,R5>           ; Save registers
                62    1A A3         28  05EB  2147         MOVC3   CXB$W_BCNT(R3),(R2),(R1) ; Copy the data
                            61          05EF
                            38    BA    05F0  2148         POPR    #^M<R3,R4,R5>           ; Restore registers
                                        05F2  2149 ;
                                        05F2  2150 ; Find next ring entry and insert data
                                        05F2  2151 ;
                52    18 A4       9A  05F2  2152         MOVZBL  CDB_B_NEXTXMT(R4),R2    ; Get next ring entry
                      18 A4       96  05F6  2153         INCB    CDB_B_NEXTXMT(R4)       ; Bump ring pointer
                      FC 8F       8A  05F9  2154         BICB    #^C<#MAX_C_XMT-1>,-     ; Modulo xmit ring size
                      18 A4           05FC  2155                 CDB_B_NEXTXMT(R4)       ;
                23 A3    52       90  05FE  2156         MOVB    R2,CXB$B_XQ_RING(R3)    ; Save ring entry number
                22 A3    57       90  0602  2157         MOVB    R7,CXB$B_XQ_SLOT(R3)    ; Save mapping slot number used
                                        0606  2158         SETBIT  R7,CDB_B_XMTMAP(R4)     ; Set mapping slot in use flag
                52    009C C442     D0  060B  2159         MOVL    CDB_L_XRINGVA(R4)[R2],R2; Get ring entry virtual address
                            62      B4  0611  2160         CLRW    XMT_W_FLAG(R2)          ; Zero the FLAG word
                   8000 8F          B0  0613  2161         MOVW    #XMT_STS_M_LAST,-       ; Init STATUS word
                      08 A2             0617  2162                 XMT_Q_STS(R2)           ;
                50    1A A3       3C  0619  2163         MOVZWL  CXB$W_BCNT(R3),R0       ; Get BYTE count
:RNG0001           3C    50       B1  061D    .1         CMPW    R0,#MIN_PKT_SIZE+XQ_C_HEADER ; Is packet at least minimum size?
:RNG0001                    07    1E  0620    .2         BGEQU   210$                    ; Br if yes, okay
:RNG0001           50    3C       D0  0622    .3         MOVL    #MIN_PKT_SIZE+XQ_C_HEADER,R0 ; Else set to minimum
:RNG0001                                0625    .4         ;
:RNG0001                                0625    .5         ;******
:RNG0001                                0625    .6         ;  The following instruction is needed to keep the code honest, that
:RNG0001                                0625    .7         ;  figures out whether the end byte is odd.
:RNG0001                                0625    .8         ;******
:RNG00C1                                0625    .9         ;
:RNG0001          1A A3       01      AA  0625    .10         BICW    #1,CXB$W_BCNT(R3)       ; Make the count look even!
:RNG0001                      50    D6  0629    .11 210$:   INCL    R0                      ; Round up by one
-4                50    FF 8F       78  062B  2168         ASHL    #-1,R0,R0               ; Convert to WORD count
                            50          062F
                06 A2       50      AE  0630  2169         MNEGW   R0,XMT_W_LEN(R2)        ; Store message length (2's complement)
              50    00C0 C447     D0  0634  2170         MOVL    CDB_L_XMT_PA(R4)[R7],R0 ; Get contiguous buffer's PA
                04 A2       50      B0  063A  2171         MOVW    R0,XMT_W_ADDR(R2)       ; Move buffer address - BA00-BA08
                50    F0 8F       78  063E  2172         ASHL    #-16,R0,R0              ; Shift down hi order address lines
                            50          0642
                02 A2       50      B0  0643  2173         MOVW    R0,XMT_W_ADDRHI(R2)     ; Insert BA16-BA21 & zero descriptor bits
                   56       52      D0  0647  2174         MOVL    R2,R6                   ; Save xmit ring entry address
                                        064A  2175         ;
```

```
;RNG0001                        064A     .1          ; Descriptor bit settings are calculated as follows.
;RNG0001                        064A     .2          ;
;RNG0001                        064A     .3          ;  If the beginning address is Even or ODD, then the END address is:
;RNG0001                        064A     .4          ;
;RNG0001                        064A     .5          ;       BEGINNING  &  LENTH, then END address is:
;RNG0001                        064A     .6          ;       ---------     -----       ---
;RNG0001                        064A     .7          ;          Even          Even        Even
;RNG0001                        064A     .8          ;          Even          Odd         Odd
;RNG0001                        064A     .9          ;          Odd           Even        Odd
;RNG0001                        064A    .10          ;          Odd           Odd         Even
;RNG0001                        064A    .11          ;
;RNG0001          50  A000 8F   B0  064A    .12          MOVW    #XMT_DSC_M_VALID!-      ; Build descriptor flag
;RNG0001                        064F    .13                   XMT_DSC_M_EOM,R0
;RNG0001              FF28  31  064F    .14          BRW     70$                     ; Continue - always even start buffers!
;RNG0001                        0652    .15
;RNG0001                        0652    .16 220$:    POPQ    R6                      ; Restore R6,R7
-14               50   01  9A  0655   2190          MOVZBL  S^#SS$_NORMAL,R0        ; Good return
                       05  0658   2191          RSB                             ; Return to caller
                           0659   2192
                           0659   2193          .DSABL  LSB
                           0659   2194
```

```
                     0659  2196          .SBTTL  RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
                     0659  2197 ;++
                     0659  2198 ; RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
                     0659  2199 ;
                     0659  2200 ; Functional description:
                     0659  2201 ;
                     0659  2202 ; The specified buffer is checked for accessibility. The buffer address and count
                     0659  2203 ; are saved in the packet. Then IPL is set to device fork IPL and if a message is
                     0659  2204 ; available the operation is completed; otherwise, the packet is queued onto
                     0659  2205 ; the waiting receive list.
                     0659  2206 ;
                     0659  2207 ; The QIO parameters for WRITES are:
                     0659  2208 ;       P1 = Address of the data buffer
                     0659  2209 ;       P2 = Size of the data buffer
                     0659  2210 ;       P5 = Optional address of the buffer to receive the source address
                     0659  2211 ;
                     0659  2212 ; Inputs:
                     0659  2213 ;
                     0659  2214 ;       R3 = IRP address
                     0659  2215 ;       R4 = PCB address
                     0659  2216 ;       R5 = UCB address
                     0659  2217 ;       R6 = CCB address
                     0659  2218 ;       R7 = Function code
                     0659  2219 ;       AP = Address of the first operation specific qio parameter
                     0659  2220 ;
                     0659  2221 ;       IPL = ASTDEL
                     0659  2222 ;
                     0659  2223 ; Outputs:
                     0659  2224 ;
                     0659  2225 ;       R0 = Status of the receive qio operation
                     0659  2226 ;       R3 = IRP address
                     0659  2227 ;       R5 = UCB ADDRESS
                     0659  2228 ;
                     0659  2229 ;       R1,R2 are destroyed.
                     0659  2230 ;--
                     0659  2231
                     0659  2232 RCV_FDT::                                       ; Read operation FDT
                     0659  2233 ;
                     0659  2234 ;  Check the request params
                     0659  2235 ;
          30 A3  B4  0659  2236          CLRW    IRP$W_BOFF(R3)                  ; Set no quota to here
       50    0C  9A  065C  2237          MOVZBL  S^#SS$_ACCVIO,R0                ; Assume access violation
   57  10 AC  D0  065F  2238          MOVL    P5(AP),R7                       ; Get address for source address
              33  13  0663  2239          BEQL    10$                             ; Br if none
                     0665  2240          IFNOWRT #RHDR_C_DATA,(R7),20$           ; Check for write access to buffer
              07  E0  066B  2241          BBS     #IRP$V_DIAGBUF,-                ; Br if diagnostic buffer given
        28 2A A3         066D  2242                  IRP$W_STS(R3),10$               ;       ...
       51    1A  9A  0670  2243          MOVZBL  #RHDR_C_LENGTH,R1               ; Get size of header buffer
              53  DD  0673  2244          PUSHL   R3                              ; Save IRP address
00000000'GF   16  0675  2245          JSB     G^EXE$ALLOCBUF                  ; Allocate header buffer
              53 8ED0  067B  2246          POPL    R3                              ; Restore IRP address
       40 50  E9  067E  2247          BLBC    R0,20$                          ; Br if allocation failure
    4C A3  52  D0  0681  2248          MOVL    R2,IRP$L_DIAGBUF(R3)            ; Save buffer address
      0080 8F  A8  0685  2249          BISW    #IRP$M_DIAGBUF,IRP$W_STS(R3) ; Indicate diag buffer present
        2A A3         0689
                     068B  2250          ASSUME  RHDR_L_DATA EQ 0
    82    0C A2  9E  068B  2251          MOVAB   RHDR_T_DATA(R2),(R2)+           ; Set address of start of data
```

```
                     068F  2252          ASSUME  RHDR_L_BUFFER EQ RHDR_L_DATA+4
    82    57    D0   068F  2253          MOVL    R7,(R2)+                ; Set user buffer address
                     0692  2254          ASSUME  RHDR_W_SIZE EQ RHDR_L_BUFFER+4
    82    51    B0   0692  2255          MOVW    R1,(R2)+                ; Save size of allocation
                     0695  2256          ASSUME  RHDR_B_TYPE EQ RHDR_W_SIZE+2
    82    13    90   0695  2257          MOVB    #DYN$C_BUFIO,(R2)+      ; Set structure type
    50    14    9A   0698  2258 10$:      MOVZBL  S^#SS$_BADPARAM,R0      ; Assume illegal size
 51    04    AC    3C   069B  2259          MOVZWL  P2(AP),R1               ; Get size
                     069F  2260
                     069F  2261
                     069F  2262
          20    13   069F  2263          BEQL    20$                     ; Br if zero - illegal
    50    6C    D0   06A1  2264          MOVL    P1(AP),R0               ; Get user buffer address
 3C A3    50    D0   06A4  2265          MOVL    R0,IRP$L_XQ_DATBUF(R3)  ; Save user VA for completion
00000000'GF    16   06A8  2266          JSB     G^EXE$READCHK           ; Check the buffer
                     06AE  2267                                          ; (No return on NO ACCESS)
 2A A3    20    A8   06AE  2268          BISW    #IRP$M_CHAINED,IRP$W_STS(R3) ; Allow data chaining
                     06B2  2269          SETIPL  UCB$B_FIPL(R5)          ; Raise IPL to lock data base
          12    10   06B6  2270          BSBB    RCV_START               ; Process the request
       06 50    E9   06B8  2271          BLBC    R0,20$                  ; Br if error
00000000'GF    17   06BB  2272          JMP     G^EXE$QIORETURN         ; Else, take normal return
                     06C1  2273
       FC7C    31   06C1  2274 20$:      BRW     ABORTIO                 ; Abort the request
```

```
                                        06C4  2276          .SBTTL  RCV_START - START RECEIVE I/O OPERATION
                                        06C4  2277 ;++
                                        06C4  2278 ; RCV_START - START RECEIVE I/O OPERATION
                                        06C4  2279 ;
                                        06C4  2280 ; Functional description:
                                        06C4  2281 ;
                                        06C4  2282 ;  Check for device active.  Receives cannot be queued to the UCB receive
                                        06C4  2283 ;  queue unless the UCB has been initialized via routine START.   They
                                        06C4  2284 ;  cannot be put in the IRP queue since this could result in non-sequential
                                        06C4  2285 ;  receive processing due to the existence of the separate receive queue.
                                        06C4  2286 ;
                                        06C4  2287 ; Inputs:
                                        06C4  2288 ;
                                        06C4  2289 ;       R3 = IRP address
                                        06C4  2290 ;       R5 = UCB address
                                        06C4  2291 ;
                                        06C4  2292 ;       IPL = FIPL
                                        06C4  2293 ;
                                        06C4  2294 ; Outputs:
                                        06C4  2295 ;
                                        06C4  2296 ;       R0 = Status of receive request
                                        06C4  2297 ;       R4 = CDB address
                                        06C4  2298 ;
                                        06C4  2299 ;       R1,R2 are destroyed.
                                        06C4  2300 ;
                                        06C4  2301 ;--
                                        06C4  2302 INACT_ERROR:
        50    20D4 8F    3C             06C4  2303          MOVZWL  #SS$_DEVINACT,R0        ; Setup return status
                         05             06C9  2304          RSB                             ; Return to caller
                                        06CA  2305
                                        06CA  2306          .ENABL  LSB
                                        06CA  2307 RCV_START::                              ; Start receive I/O operation
                               04   E1  06CA  2308          BBC     #UCB$V_XQ_RUN,-         ; Br if UCB is not in RUN mode
                            F5 68 A5    06CC  2309                  UCB$W_DEVSTS(R5),INACT_ERROR
                                        06CF  2310
:RNG0001                54    00C8 C5   D0  06CF    .1          MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2                                 39   13  06D4  2313          BEQL    40$                     ; Br if none
                                   01   E1  06D6  2314          BBC     #CDB_STS_V_RUN,-        ; Br if QNA not running
                              E8 0246 C4    06D8  2315                  CDB_B_STS(R4),INACT_ERROR
                        51    00A0 C5   9E  06DC  2316          MOVAB   UCB$Q_XQ_RCVMSG(R5),R1  ; Get address of UCB received messages
                                        06E1  2317 ;
                                        06E1  2318 ; If running in SHARED mode, then use the listheads in the SHR_
                                        06E1  2319 ; data structure.
                                        06E1  2320 ;
                                   03   E1  06E1  2321          BBC     #UCB$V_XQ_SHARE,-       ; Br if UCB is NOT SHARED
                            09 68 A5    06E3  2322                   UCB$W_DEVSTS(R5),5$    ;
                                        06E6  2323 ;
                                        06E6  2324 ; Try to find a match on PID/CHAN
                                        06E6  2325 ;
                                  0027  30  06E6  2326          BSBW    MATCH_SHR               ; Try to find shared user
                                    D9  12  06E9  2327          BNEQ    INACT_ERROR             ; Br if none - inactive user
                          51    18 A1   9E  06EB  2328          MOVAB   SHR_Q_RCVMSG(R1),R1     ; Get address of received messages
                                        06EF  2329 ;
                                        06EF  2330 ; Check to see if message is available
                                        06EF  2331 ;
                             52   91   0F  06EF  2332 5$:       REMQUE  @(R1)+,R2               ; Dequeue a received message
                                        06F2  2333                                          ; ..bump pointer to end of list pointer
```

```
                 05   1D  06F2  2334          BVS     10$                     ; Br if none
                          06F4  2335 ;
                          06F4  2336 ; Complete receive with available message
                          06F4  2337 ;
                1393   30  06F4  2338          BSBW    FINISH_RCV_IO           ; Complete the receive
                  13   11  06F7  2339          BRB     30$                     ; And exit
                          06F9  2340 ;
                          06F9  2341 ; Queue IRP for future message arivial unless IO$M_NOW specified
                          06F9  2342 ;
          0A 20 A3    06  E1  06F9  2343 10$:    BBC     #IO$V_NOW,IRP$W_FUNC(R3),20$ ; Br if not READ NOW
           50   08F0 8F  3C  06FE  2344          MOVZWL  #SS$_ENDOFFILE,R0       ; Set no message status
                144A  30  0703  2345          BSBW    IO_DONE                 ; Complete the I/O
                  04  11  0706  2346          BRB     30$                     ; And exit
                          0708  2347 ;
                          0708  2348 ; Queue the IRP to UCB receive wait queue
                          0708  2349 ;
                          0708  2350          ASSUME  UCB$Q_XQ_RCVREQ EQ UCB$Q_XQ_RCVMSG+8
                          0708  2351          ASSUME  SHR_Q_RCVREQ EQ SHR_Q_RCVMSG+8
          08 B1    63   0E  0708  2352 20$:    INSQUE  (R3),@8(R1)             ; Put packet on waiting list
             50    01   9A  070C  2353 30$:    MOVZBL  S^#SS$_NORMAL,R0        ; Set QIO status return
                   05   0F  070F  2354 40$:    RSB                             ; Return to caller
                          0710  2355          .DSABL  LSB
```

```
                     0710  2357          .SBTTL   SUBROUTINES TO FIND SHR DATA STRUCTURE
                     0710  2358 ;+
                     0710  2359 ; Subroutine to find SHR data structure for user
                     0710  2360 ;
                     0710  2361 ; Inputs:
                     0710  2362 ;       R3 = Address of IRP
                     0710  2363 ;       R5 = UCB address
                     0710  2364 ;
                     0710  2365 ; Outputs:
                     0710  2366 ;       R1 = Address if SHR data structure if match
                     0710  2367 ;       R0 is destroyed.
                     0710  2368 ;       Z-Bit set then match.
                     0710  2369 ;       Z-Bit clear then no match.
                     0710  2370 ;-
                     0710  2371
                     0710  2372 MATCH_SHR:                                  ; Try to find shared user
51   00C4 C5   DO    0710  2373          MOVL    UCB$L_XQ_DEFUSR(R5),R1     ; Get address of default user
          04   13    0715  2374          BEQL    10$                        ; Br if no default user
          1C   10    0717  2375          BSBB    CHECK_SHR                  ; Check for match
          19   13    0719  2376          BEQL    40$                        ; Br if match
50   0098 C5   9E    071B  2377 10$:     MOVAB   UCB$Q_XQ_SHARE(R5),R0      ; Save address of listhead
     51   50   DO    0720  2378          MOVL    R0,R1                      ; Copy listhead address
                     0723  2379          ASSUME  SHR_L_QFL EQ 0
     51   61   DO    0723  2380 20$:     MOVL    (R1),R1                    ; Get next in list
     50   51   D1    0726  2381          CMPL    R1,R0                      ; Back to start of list?
          06   13    0729  2382          BEQL    30$                        ; Br if yes - no pid/chan match
          08   10    072B  2383          BSBB    CHECK_SHR                  ; Check for match
          F4   12    072D  2384          BNEQ    20$                        ; Br if none
          03   11    072F  2385          BRB     40$                        ; Return in success
     50   50   DO    0731  2386 30$:     MOVL    R0,R0                      ; Return match failure
          05   0734  2387 40$:     RSB
                     0735  2388
                     0735  2389 ;+
                     0735  2390 ; Subroutine to check if PID and SHR data base match up
                     0735  2391 ;
                     0735  2392 ; Inputs:
                     0735  2393 ;       R1 = Address of SHR
                     0735  2394 ;       R3 = Address of IRP
                     0735  2395 ;
                     0735  2396 ; Outputs:
                     0735  2397 ;       Z-Bit set then match.
                     0735  2398 ;       Z-Bit clear then no match.
                     0735  2399 ;-
                     0735  2400
                     0735  2401 CHECK_SHR:                                  ; Check for match with SHR data base
          0C A3   D5 0735  2402          TSTL    IRP$L_PID(R3)              ; Is this an Internal IRP user?
             09   19 0738  2403          BLSS    10$                        ; Br if yes, only one allowed per UCB
                     073A  2404 ;
                     073A  2405 ; Normal QIO user
                     073A  2406 ;
OC A1     0C A3   D1 073A  2407          CMPL    IRP$L_PID(R3),SHR_L_PID(R1) ; PIDs match?
             0F   12 073F  2408          BNEQ    30$                        ; Br if no - try for next
             08   11 0741  2409          BRB     20$                        ; Else, continue checks
                     0743  2410 ;
                     0743  2411 ; Internal IRP user
                     0743  2412 ;
          00B8 C5   D1 0743  2413 10$:     CMPL    UCB$L_XQ_PID(R5),SHR_L_PID(R1) ; Is this the Internal user?
```

```
                0C A1          0747
                   05    12    0749  2414              BNEQ    30$                                 ; Br if not
          10 A1    28 A3 B1    074B  2415 20$:         CMPW    IRPSW_CHAN(R3),SHR_W_CHAN(R1) ; Channels match?
                   05          0750  2416 30$:         RSB                                         ; Return to caller
                               0751  2417
```

```
                                0751  2419          .SBTTL  ALT_START - ALTERNATE START I/O ROUTINE
                                0751  2420 ;++
                                0751  2421 ; ALT_START - ALTERNATE STARTIO I/O ROUTINE
                                0751  2422 ;
                                0751  2423 ; Functional description:
                                0751  2424 ;
                                0751  2425 ; This routine is called by the Executive to pass an "internal" IRP
                                0751  2426 ; to the driver.  "Internal" IRP's are those not built via QIO.
                                0751  2427 ; These IRPs are used by higher level software used to request I/O and
                                0751  2428 ; should not be confused with the IRPs built and passed by the
                                0751  2429 ; Transport layer to NSP.  The action here is to setup the IRP fields
                                0751  2430 ; as if the packet had been processed by the FDT routines.
                                0751  2431 ;
                                0751  2432 ; Inputs:
                                0751  2433 ;
                                0751  2434 ;       R3 = IRP address
                                0751  2435 ;       R5 = UCB address
                                0751  2436 ;
                                0751  2437 ;       All pertinent fields of the IRP are assumed to be valid.
                                0751  2438 ;
                                0751  2439 ;       IPL = FIPL
                                0751  2440 ;
                                0751  2441 ; Implicit inputs:
                                0751  2442 ;
                                0751  2443 ;       IRP$L_SVAPTE(R3) = System VIRTUAL address (not physical PTE address)
                                0751  2444 ;
                                0751  2445 ; Outputs:
                                0751  2446 ;
                                0751  2447 ;       R0-R5 may be garbage
                                0751  2448 ;--
                                0751  2449
                                0751  2450 ALT_START::                              ; Accept an "internal" IRP
                      01    E0  0751  2451          BBS     #IRP$V_FUNC,-           ; If BS then read function
                  2D 2A A3        0753  2452                  IRP$W_STS(R3),10$
                   00CE C5    B0  0756  2453          MOVW    UCB$W_XQ_PROTYP(R5),-   ; MUST be a non-promiscous user
                      3A A3        0759  2454                  IRP$W_XQ_PROTYP(R3)     ;
              52      2C A3    D0  075C  2455          MOVL    IRP$L_SVAPTE(R3),R2     ; Get address of start of data
          51  62      0E    C3  0760  2456          SUBL3   #XQ_C_HEADER,(R2),R1    ; Get the xmit buffer address
              51      52    C2  0764  2457          SUBL    R2,R1                   ; Form offset to start of data
            18 A2     51    B0  0767  2458          MOVW    R1,CXB$W_BOFF(R2)       ; Store offset in CXB
            0A A2     1B    90  076B  2459          MOVB    #DYN$C_CXB,CXB$B_TYPE(R2) ; Set structure type to CXB
               00DC   C5    91  076F  2460          CMPB    UCB$B_XQ_PRO(R5),-      ;% Point-to-point mode?
                      00        0773  2461                  #NMA$C_LINPR_POI
                      03    12  0774  2462          BNEQ    5$                      ;% If so,
                   40 A3    7C  0776  2463          CLRQ    IRP$Q_STATION(R3)       ;% Pick up destination from SHR block
                      53    DD  0779  2464 5$:      PUSHL   R3                      ; Save IRP address
                    FC0B    30  077B  2465          BSBW    XMT_START               ; Start transmit operation
                      53 8ED0   077E  2466          POPL    R3                      ; Restore IRP address
                      14    11  0781  2467          BRB     30$                     ; Continue
                                0783  2468
              52      2C A3    D0  0783  2469 10$:     MOVL    IRP$L_SVAPTE(R3),R2     ; Get address of input buffer
                      0B    13  0787  2470          BEQL    20$                     ; Br if none
;RNG0001       54    00C8 C5    D0  0789     ,1      MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2                    2C A3    D4  078E  2473          CLRL    IRP$L_SVAPTE(R3)        ; Make sure SVAPTE is cleared
                                0791  2474 ;
                                0791  2475 ; The driver must be prepared to process chained buffers returned from
                                0791  2476 ; the higher levels.
```

```
                     0791  2477 ;
                     0791  2478 ;&&15$: PUSHL   CXB$L_LINK(R2)          ; Save address of next in link
                     0791  2479 ;&&      CLRL    CXB$L_LINK(R2)          ; Clear the link cell
          0CA1   30  0791  2480          BSBW    ADDRCVLIST              ; Else, add it to the receive list
                     0794  2481 ;&&      POPL    R2                      ; Get back address of next in chain
                     0794  2482 ;&&      BNEQ    15$                     ; Br if more to return
                     0794  2483
          FF33   30  0794  2484 20$:     BSBW    RCV_START               ; Start receive operation
        01 50    E9  0797  2485 30$:     BLBC    R0,40$                  ; Br if error
                 05  079A  2486          RSB                             ; Return to caller
                     079B  2487
          13B2   31  079B  2488 40$:     BRW     IO_DONE                 ; Post the I/O request in error
```

```
079E  2490          .SBTTL  SETMODE_FDT - SET MODE I/O OPERATION FDT DISPATCH ROUTINE
079E  2491 ;++
079E  2492 ; SETMODE_FDT - SET MODE FDT PROCESSING
079E  2493 ;
079E  2494 ; Functional description:
079E  2495 ;
079E  2496 ; This is the fdt routine for setmode functions.
079E  2497 ; There are three functions based on subfunction modifier bit.
079E  2498 ;
079E  2499 ; NOTE: That there is no difference on a request to shutdown a line or
079E  2500 ;       a circuit. However, a request to startup a circuit is ignored
079E  2501 ;       completely.
079E  2502 ;
079E  2503 ; The QIO parameters for SETMODE are:
079E  2504 ;
079E  2505 ;       P2 = Optional address of buffer descriptor for extended characteristics
079E  2506 ;
079E  2507 ;
079E  2508 ; The Subfunction modifiers are as follows:
079E  2509 ;
079E  2510 ; 1)     CHANGE MODE -- NO MODIFIER BIT.
079E  2511 ;       This function is done in the STARTIO routine. Control is passed to
079E  2512 ;       EXE$SETMODE to validate the new mode buffer and queue the packet.
079E  2513 ;
079E  2514 ; 2)     INITIALIZE THE UNIT -- IO$M_STARTUP SET.
079E  2515 ;       This function is done partially here and the remainder in STARTIO.
079E  2516 ;       The action here is to pick up the user buffered I/O quota.  The quota
079E  2517 ;       taken from the user is in IRP$W_BOFF. This value will be the IOSB+2 value
079E  2518 ;       at I/O done. The mailbox is enabled and a receive is started.
079E  2519 ;
079E  2520 ; 3)     SHUTDOWN UNIT -- IO$M_SHUTDOWN SET.
079E  2521 ;       This function shuts down the unit and optionally resets the mode.
079E  2522 ;       A CANCEL I/O is performed, all outstanding I/O is completed, the
079E  2523 ;       message blocks are all returned and the unit is left in an idle
079E  2524 ;       state. This function cannot be done here and the FDT processing is
079E  2525 ;       that of all SETMODE operations.
079E  2526 ;
079E  2527 ; 4)     ATTENTION AST -- IO$M_ATTNAST SET.
079E  2528 ;       This function sets up an AST to be delivered when a change of
079E  2529 ;       status occurs on the QNA.
079E  2530 ;
079E  2531 ;
079E  2532 ; Inputs:
079E  2533 ;
079E  2534 ;       R3 = IRP ADDESS
079E  2535 ;       R4 = PCB ADDRESS
079E  2536 ;       R5 = UCB ADDRESS
079E  2537 ;       R6 = CCB ADDRESS
079E  2538 ;       R7 = FUNCTION CODE
079E  2539 ;       AP = ADDRESS OF THE FIRST QIO PARAMETER
079E  2540 ;
079E  2541 ;       IPL = ASTDEL
079E  2542 ;
079E  2543 ; Outputs:
079E  2544 ;
079E  2545 ;       R3 = IRP ADDRESS
079E  2546 ;       R4 = PCB ADDRESS
```

```
                     079E  2547 ;       R5 = UCB ADDRESS
                     079E  2548 ;
                     079E  2549 ;       R0-R2,R6 are destroyed.
                     079E  2550 ;
                     079E  2551 ;--
                     079E  2552
                     079E  2553 SETMODE_FDT::                                ; SET MODE FDT processing
          2C A3   D4 079E  2554          CLRL    IRP$L_SVAPTE(R3)            ; Set no buffered packet
          38 A3   7C 07A1  2555          CLRQ    IRP$L_MEDIA(R3)             ; Reset mode data area
       0094 C3    D4 07A4  2556          CLRL    IRP$L_XQ_SETUP(R3)          ; Indicate no SETUP buffer yet
 50    0084 8F    3C 07A8  2557          MOVZWL  #SS$_DEVOFFLINE,R0          ; Assume unit if offline
          04      E0 07AD  2558          BBS     #UCB$V_ONLINE,-             ; Br if unit online
    03 64 A5           07AF  2559                  UCB$W_STS(R5),5$            ;
        FBBB      31 07B2  2560          BRW     ABORTIO                     ; Else, abort the I/O request
                     07B5  2561
  57    20 A3   B0 07B5  2562 5$:      MOVW    IRP$W_FUNC(R3),R7           ; Get entire function code
  5C 57    08   E1 07B9  2563          BBC     #IO$V_ATTNAST,R7,30$        ; Br if not attention AST
                     07BD  2564 ;
                     07BD  2565 ; User is requesting an attention AST
                     07BD  2566 ;
 57   00C0 C5   DE 07BD  2567          MOVAL   UCB$L_XQ_AST(R5),R7         ; Get address of AST list
  00000000'GF    16 07C2  2568          JSB     G^COM$SETATTNAST            ; Set up attention AST
                     07C8  2569
                     07C8  2570          ASSUME  UCB$V_XQ_INITED  EQ  0
    12 68 A5   E9 07C8  2571          BLBC    UCB$W_DEVSTS(R5),10$        ; Br if protocol not active
                     07CC  2572
 51   00A0 C5   9E 07CC  2573          MOVAB   UCB$Q_XQ_RCVMSG(R5),R1      ; Check for empty receive list
      61   51   D1 07D1  2574          CMPL    R1,(R1)                     ; Empty?
           08   13 07D4  2575          BEQL    10$                         ; Br if YES, no need to inform user
           53   DD 07D6  2576          PUSHL   R3                          ; Save IRP address
         2288   30 07D8  2577          BSBW    POKE_USER                   ; Inform user
           53 8ED0  07DB  2578          POPL    R3                          ; Restore IRP address
    50    01   9A 07DE  2579 10$:      MOVZBL  S^#SS$_NORMAL,R0            ; Set success
 51    44 A5   D0 07E1  2580          MOVL    UCB$L_DEVDEPEND(R5),R1      ; Get device dependent information
00000000'GF    17 07E5  2581 20$:      JMP     G^EXE$FINISHIO              ; Complete the I/O
                     07EB  2582 ;
                     07EB  2583 ; On a circuit request,
                     07EB  2584 ;       If this is a shutdown then perform a $CANCEL and clear the RUN flag.
                     07EB  2585 ;       If this is a startup then set the RUN flag.
                     07EB  2586 ;
 14 57    07   E1 07EB  2587 25$:      BBC     #IO$V_SHUTDOWN,R7,27$       ; Br if not a shutdown request
                     07EF  2588          SETIPL  UCB$B_FIPL(R5)              ; Sync acces to UCB & CDB
                     07F3  2589          CLRBIT  #UCB$V_XQ_RUN,-             ; Clear the RUN flag
                     07F3  2590                  UCB$W_DEVSTS(R5)
 52    28 A3   3C 07F8  2591          MOVZWL  IRP$W_CHAN(R3),R2           ; Get channel number
                     07FC  2592          ASSUME  CAN$C_CANCEL EQ 0
           58   D4 07FC  2593          CLRL    R8                          ; Set $CANCEL function
         1B14   30 07FE  2594          BSBW    CANCEL                      ; Perform a CANCEL
           DB   11 0801  2595          BRB     10$                         ; Complete the request
                     0803  2596
 D7 57    06   E1 0803  2597 27$:      BBC     #IO$V_STARTUP,R7,10$        ; Br if not a startup request
                     0807  2598          CLRBIT  #XM$V_ERR_START,UCB$L_DEVDEPEND(R5) ;% Clear start error flag
    0060 8F    A8 080C  2599          BISW    #UCB$M_XQ_START!UCB$M_XQ_STACK,- ,% Set start and stack states
      68 A5           0810  2600                  UCB$W_DEVSTS(R5)            ;%
                     0812  2601          SETBIT  #UCB$V_XQ_RUN,-             ; Set the RUN flag
                     0812  2602                  UCB$W_DEVSTS(R5)
           C5   11 0817  2603          BRB     10$                         ; Complete the request
```

```
                                0819  2604 ;
                                0819  2605 ; For everthing except Attention ASTs we must make sure CDB is present
                                0819  2606 ; and we must verify the P2 buffer.
                                0819  2607 ;
;RNG0001                 00C8 C5   D5   0819     .1 30$:    TSTL    UCB$L_XQ_CDB(R5)          ; Is CDB there?
-2                            0E   12   081D  2610          BNEQ    35$                       ; Br if yes
                            1691   30   081F  2611          BSBW    ALLOC_CDB                 ; Else, allocate CDB
                          08 50    E8   0822  2612          BLBS    R0,35$                    ; Br if successful
                     50   0124 8F   3C   0825  2613          MOVZWL  #SS$_INSFMEM,R0           ; Set error return
                            FB13   31   082A  2614 33$:     BRW     ABORTIO                   ; Return error
                                   082D  2615
                            054D   30   082D  2616 35$:     BSBW    GET_CHAR_BUF              ; Get P2 characteristics
                          F7 50    E9   0830  2617          BLBC    R0,33$                    ; Br if error - Abort I/O
                     52   F841 CF   9E   0833  2618          MOVAB   LINE_PARAM_WO,R2          ; Assume the line parameters
                       05 57   09    E0   0838  2619          BBS     #IO$V_CTRL,R7,36$         ; Br if line request
                     52   F8BE CF   9E   083C  2620          MOVAB   CIRCUIT_PARAM,R2          ; Else, use the circuit parameters
                            1DD5   30   0841  2621 36$:     BSBW    VALIDATE_P2               ; Validate the P2 parameters
                          9E 50    E9   0844  2622          BLBC    R0,20$                    ; Br if error
                        A0 57   09    E1   0847  2623          BBC     #IO$V_CTRL,R7,25$         ; Br if not a LINE request
                            1CC3   30   084A  2624          SETIPL  UCB$B_FIPL(R5)            ; Sync access to UCB's
                            1CC3   30   084E  2625          BSBW    SAV_MULTI                 ; Save the multicast address list
                      0090 C3      D4   0852  2626          CLRL    IRP$L_XQ_SHR(R3)                  ; Assume exclusive user
                           03      E1   0856  2627          BBC     #UCB$V_XQ_SHARE,-         ; Br if not a SHARED user
                        16 68 A5        0858  2628                  UCB$W_DEVSTS(R5),40$      ;
                                        085B  2629 ;
                                        085B  2630 ;       Allow the shared user to change the destination node with which
                                        085B  2631 ;       it is communicating.
                                        085B  2632 ;
                            FEB2   30   085B  2633          BSBW    MATCH_SHR                 ; Else, try to find shared user
                              11   12   085E  2634          BNEQ    40$                       ; Br if none found, skip it
                     0090 C3    51   D0   0860  2635          MOVL    R1,IRP$L_XQ_SHR(R3)       ; Save the SHR data structure address
                          12 A1    D0   0865  2636          MOVL    SHR_G_DEST(R1),-          ; Save the current destination user
                        00D0 C5         0868  2637                  UCB$G_XQ_DES(R5)          ;  address in the UCB
                          16 A1    B0   086B  2638          MOVW    SHR_G_DEST+4(R1),-        ;       ...
                        00D4 C5         086E  2639                  UCB$G_XQ_DES+4(R5)        ;       ...
                                        0871  2640 ;
                                        0871  2641 ;       Now we will set the parameters given in the setmode request. But,
                                        0871  2642 ;       first if the DEQNA controller is inited we will use the current
                                        0871  2643 ;       hardware settings for the defaults.
                                        0871  2644 ;
;RNG0001                 51   00C8 C5   D0   0871     .1 40$:    MOVL    UCB$L_XQ_CDB(R5),R1       ; Get CDB address
-2                                      0876  2647          ASSUME  CDB_STS_V_INITED EQ 0
                          13 0246 C1    E9   0876  2648          BLBC    CDB_B_STS(R1),45$         ; Br if controller not enabled, use
                                        087B  2649                                            ;  the fixed defaults
                     51   0249 C1    9E   087B  2650          MOVAB   CDB_B_SETPRM(R1),R1       ; Get address of setable parameters
                     52   00E1 C5    9E   0880  2651          MOVAB   UCB$B_XQ_CDBPRM(R5),R2    ; Get address of UCB parameters
                          50   01    9A   0885  2652          MOVZBL  #UCB$C_XQ_CDBPRM,R0       ; Get size of parameters to move
                          82   81    90   0888  2653 42$:     MOVB    (R1)+,(R2)+               ; Store CDB parameters into UCB
                            FA 50    F5   088B  2654          SOBGTR  R0,42$                    ; Loop if more
                     52   F7E6 CF    9E   088E  2655 45$:     MOVAB   LINE_PARAM_WO,R2          ; Get address of verification table
                            1DEE   30   0893  2656          BSBW    CHANGE_PARAM              ; Change the parameters
                     51   0090 C3    D0   0896  2657          MOVL    IRP$L_XQ_SHR(R3),R1       ; Get the SHR structure address
                              0C    13   089B  2658          BEQL    50$                       ; Br if not present, skip it
                          00D0 C5    D0   089D  2659          MOVL    UCB$G_XQ_DES(R5),-        ; Else, reset the destination user
                            12 A1         08A1  2660                  SHR_G_DEST(R1)            ;  address into the SHR structure
                          00D4 C5    B0   08A3  2661          MOVW    UCB$G_XQ_DES+4(R5),-      ;       ...
                            16 A1         08A7  2662                  SHR_G_DEST+4(R1)          ;       ...
```

```
                                08A9  2663
                     0E 57  07  E1  08A9  2664 50$:    BBC     #IO$V_SHUTDOWN,R7,60$    ; Br if not shutdown request
                                08AD  2665 ;
                                08AD  2666 ; Shutdown protocol request
                                08AD  2667 ;
                                08AD  2668          ASSUME  UCB$V_XQ_INITED EQ 0
                       03 68 A5 E8  08AD  2669          BLBS    UCB$W_DEVSTS(R5),55$     ; Br if still inited
                           FF2A 31  08B1  2670          BRW     10$                      ; Else, complete I/O request now
:RNG0001             21 A3  03  90  08B4     ,1 55$:   MOVB    #XQ_FC_V_STOP,IRP$B_XQ_FUNC(R3) ; Set internal function code
-1                       03F8 31  08B8  2672          BRW     QUEPKT                   ; Queue request to QNA
                                08BB  2673
                     03 57  06  E0  08BB  2674 60$:    BBS     #IO$V_STARTUP,R7,80$     ; Br if startup function
                           0102 31  08BF  2675 70$:    BRW     180$                     ; Else, must be change mode
                                08C2  2676 ;
                                08C2  2677 ; Startup protocol request
                                08C2  2678 ;
                     21 A3  00  90  08C2  2679 80$:    MOVB    #XQ_FC_V_INIT,IRP$B_XQ_FUNC(R3) ; Insert internal function code
                                08C6  2680 ;
                                08C6  2681 ;       If the UCB is already initialized for SHARED use, then we will
                                08C6  2682 ;       check to make sure that the SHR struture exists. If it does then
                                08C6  2683 ;       the share structure must be active, by definition.
                                08C6  2684 ;
                       0090 C3  D5  08C6  2685          TSTL    IRP$L_XQ_SHR(R3)                 ; Was the SHR structure present
                            F3  12  08CA  2686          BNEQ    70$                      ; Br if yes, already started
                            03  E0  08CC  2687          BBS     #UCB$V_XQ_SHARE,-        ; Br if SHARED UCB, ingore status
                     61 68 A5        08CE  2688                  UCB$W_DEVSTS(R5),125$    ; ..make user a shared user
                                08D1  2689          ASSUME  UCB$V_XQ_INITED EQ 0
                     EA 68 A5  E8  08D1  2690          BLBS    UCB$W_DEVSTS(R5),70$     ; Br if already started
                                08D5  2691                                               ; ..check multicast address list
                            02  E1  08D5  2692          BBC     #UCB$V_XQ_PROTYP,-       ; Br if no protocol specified
                     43 68 A5        08D7  2693                  UCB$W_DEVSTS(R5),100$    ; ..error
                                08DA  2694 ;
                                08DA  2695 ; Check if protocol type is to be shared
                                08DA  2696 ;
                           03   91  08DA  2697          CMPB    #NMA$C_ACC_EXC,-         ; Is this PROTOCOL TYPE for exclusive
                       00D8 C5      08DC  2698                   UCB$B_XQ_ACC(R5)         ;  use?
                            51  12  08DF  2699          BNEQ    125$                     ; Br if not
                                08E1  2700 ;
                                08E1  2701 ; Check protocol type for uniqueness
                                08E1  2702 ;
                   51   00CE C5  3C  08E1  2703          MOVZWL  UCB$W_XQ_PROTYP(R5),R1   ; Get protocol type
                                08E6  2704 ;
                                08E6  2705 ; For a user wishing to run in promiscuous mode, the requirement is that
                                08E6  2706 ; there be no other promiscuous users running. For a non-promiscuous user,
                                08E6  2707 ; there must be no other users running with the same protocol type.
                                08E6  2708 ;
                 00002A9B'EF   9E  08E6  2709          MOVAB   MATCH_PROTYP,R0          ; Get address of Action routine
                            50      08EC
                                08ED  2710                                               ;  assume non-PROMISCUOUS user
                                08ED  2711
                                08ED  2712          ASSUME  NMA$C_STATE_ON EQ 0
                                08ED  2713          ASSUME  NMA$C_STATE_OFF EQ 1
                                08ED  2714
                   1B 00DE C5  E8  08ED  2715          BLBS    UCB$B_XQ_PRM(R5),90$     ; Br if NOT a PROMISCUOUS user
                                08F2  2716 ;
                                08F2  2717 ; The promiscuous user must have PHY_IO privilege
                                08F2  2718 ;
```

```
                00002ABD'EF   9E  08F2  2719          MOVAB   MATCH_PROMTYP,R0         ; Get address of Action routine
                        50        08F8
                                  08F9  2720          IFPRIV  PHY_IO,90$              ; If user has privilege, then okay
                      1C31    30  08FF  2721          BSBW    RES_MULTI               ; Else, Restore original multicast list
                   50   24    3C  0902  2722          MOVZWL  #SS$_NOPRIV,R0          ; Return error - NOPRIV
                51 0B18 8F    3C  0905  2723          MOVZWL  #NMA$C_PCLI_PRM,R1      ; Return the bad parameter
                      FED8    31  090A  2724          BRW     20$                     ; Finish the I/O request
                                  090D  2725 90$:     PUSHQ   R4                      ; Save PCB, UCB addresses
:RNG0001        54 00C8 C5    D0  0910     .1         MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2                      60    16  0915  2728          JSB     (R0)                    ; Try to find exact match
                                  0917  2729          POPQ    R4                      ; Restore PCB, UCB addresses
                        22 50   E9  091A  2730          BLBC    R0,130$                 ; Br if none found - okay
                                  091D  2731 ;
                                  091D  2732 ;        Bad protocol type
                                  091D  2733 ;
                51 0B0E 8F    3C  091D  2734 100$:    MOVZWL  #NMA$C_PCLI_PTY,R1      ; Return bad parameter code
                           05   11  0922  2735          BRB     120$                    ; Finish error reporting
                                  0924  2736 ;
                                  0924  2737 ;        Bad quota calculated
                                  0924  2738 ;
                51 0451 8F    3C  0924  2739 110$:    MOVZWL  #NMA$C_PCLI_BFN,R1      ; Return bad parameter code
                   50   14    9A  0929  2740 120$:    MOVZBL  S^#SS$_BADPARAM,R0      ; Set error return
                      1C04    30  092C  2741 123$:    BSBW    RES_MULTI               ; Restore original multicast list
                      FEB3    31  092F  2742          BRW     20$                     ; Finish the I/O request
                                  0932  2743 ;
                                  0932  2744 ; Shared protocol type - look for same protocol type in other UCB.
                                  0932  2745 ;
                        0139    30  0932  2746 125$:    BSBW    SHR_UCB                 ; Find other UCB in user or make this
                                  0935  2747                                          ;  UCB shareable
                        F4 50   E9  0935  2748          BLBC    R0,123$                 ; Br on error
                                  0938  2749 ;
                                  0938  2750 ; We must now check if UCB is already inited.
                                  0938  2751 ;
                                  0938  2752          ASSUME  UCB$V_XQ_INITED  EQ  0
                     11 68 A5     E9  0938  2753          BLBC    UCB$W_DEVSTS(R5),135$   ; Br if this UCB is NOT inited
                                  093C  2754                                          ; skip quota taking, already done
                         008C     31  093C  2755          BRW     185$                    ; Else, compute multicast list
                                  093F  2756
                                  093F  2757 130$:    SETIPL  #IPL$_ASTDEL            ; Reset IPL to ASTDEL
                                  0942  2758 ;
                                  0942  2759 ; Take quota needed
                                  0942  2760 ;
                          0097    30  0942  2761          BSBW    TAKE_QUOTA              ; Take quota from user
                        DC 50     E9  0945  2762          BLBC    R0,110$                 ; Br if error
                00CC C5    57     B0  0948  2763          MOVW    R7,UCB$W_XQ_QUOTA(R5)   ; Save quota in UCB
                                  094D  2764 135$:    SETBIT  #UCB$V_XQ_INITED,-      ; Indicate unit is initialized
                                  094D  2765                  UCB$W_DEVSTS(R5)        ;       ...
                                  0952  2766          SETIPL  UCB$B_FIPL(R5)          ; Sync access to UCB and CDB
                   0A 00DE C5     E8  0956  2767          BLBS    UCB$B_XQ_PRM(R5),140$   ; Br if NOT a PROMISCUOUS user
:RNG0001        54   00C8 C5     D0  095B     .1         MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2              0210 C4    55     D0  0960  2770          MOVL    R5,CDB_L_PRMUSER(R4)    ; Store promiscuous user's address
                                  0965  2771 ;
                                  0965  2772 ; Pre-allocate all needed receive buffers, if the CDB is not initialized yet!
                                  0965  2773 ; The buffers are immediately deallocated, but this pre-allocation will allow
                                  0965  2774 ; the pool to grow if necessary! This must be done here, before we run on the
                                  0965  2775 ; interrupt stack.
                                  0965  2776 ;
```

```
                     0965  2777 140$:   PUSHQ   R3                          ; Save R3, R4
51   00DA C5   3C    0968  2778         MOVZWL  UCB$W_XQ_BSZ(R5),R1          ; Get size of receive buffer
54   00D6 C5   3C    096D  2779         MOVZWL  UCB$W_XQ_HBQ(R5),R4          ; Get device buffer quota
     54   51   C6    0972  2780         DIVL    R1,R4                        ; Compute number of buffers to allocate
          54   D6    0975  2781         INCL    R4                           ; Plus one extra
          7E   D4    0977  2782         CLRL    -(SP)                        ; End of list marker
00000000'GF   16    0979  2783 145$:   JSB     G^EXE$ALONONPAGED            ; Allocate the memory
       0E 50  E9    097F  2784         BLBC    R0,150$                      ; Br on error
 08 A2   51   B0    0982  2785         MOVW    R1,IRP$W_SIZE(R2)            ; Save size of buffer
 0A A2   1B   90    0986  2786         MOVB    #DYN$C_CXB,IRP$B_TYPE(R2)    ; Set structure type
     7E   52   D0    098A  2787         MOVL    R2,-(SP)                     ; Save buffer address
       E9 54   F5    098D  2788         SOBGTR  R4,145$                      ; Loop if more to allocate
                     0990  2789
     50   8E   D0    0990  2790 150$:   MOVL    (SP)+,R0                     ; Get buffer address
          08   13    0993  2791         BEQL    155$                         ; Br if end of list
00000000'GF   16    0995  2792         JSB     G^EXE$DEANONPAGED            ; Deallocate the block
          F3   11    099B  2793         BRB     150$                         ; Try for more
                     099D  2794 155$:   POPQ    R3                           ; Restore R3, R4
                     09A0  2795
        1AF0   30    09A0  2796         BSBW    ADD_MULTI                    ; Compile a new multicast address list
      33 50   E8    09A3  2797         BLBS    R0,190$                      ; Br if all okay
                     09A6  2798
                     09A6  2799         ASSUME  NMA$C_STATE_ON EQ 0
                     09A6  2800         ASSUME  NMA$C_STATE_OFF EQ 1
                     09A6  2801
 04 00DE C5   E8    09A6  2802         BLBS    UCB$B_XQ_PRM(R5),160$        ; Br if NOT a PROMISCUOUS user
    0210 C4   D4    09AB  2803         CLRL    CDB_L_PRAUSER(R4)            ; Else, clear the PROMISCUOUS user addr
                     09AF  2804 160$:   CLRBIT  #UCB$V_XQ_INITED,-           ; Indicate unit is not initialized
                     09AF  2805                 UCB$W_DEVSTS(R5)          ;       ...
 20 A6   57   C0    09B4  2806         ADDL    R7,JIB$L_BYTCNT(R6)          ; Restore quota
 24 A6   57   C0    09B8  2807         ADDL    R7,JIB$L_BYTLM(R6)           ; ..and byte limit
51   0B0F 8F   3C    09BC  2808 170$:   MOVZWL  #NMA$C_PCLI_MCA,R1           ; Indicate bad multicast address
       FF65   31    09C1  2809         BRW     120$                         ; Return in error
                     09C4  2810 ;
                     09C4  2811 ; Change mode request - might have to reset multicast address list
                     09C4  2812 ;
                     09C4  2813 180$:   ASSUME  UCB$V_XQ_INITED  EQ  0
 03 68 A5   E8    09C4  2814         BLBS    UCB$W_DEVSTS(R5),185$        ; Br if unit inited
     FE13   31    09C8  2815         BRW     10$                          ; Else, success
                     09CB  2816
                     09CB  2817 185$:   SETIPL  UCB$B_FIPL(R5)               ; Sync access to UCB and CDB
        1AC1   30    09CF  2818         BSBW    ADD_MULTI                    ; Compile global multicast address list
      E7 50   E9    09D2  2819         BLBC    R0,170$                      ; Br if error
21 A3   06   90    09D5  2820         MOVB    #XQ_FC_V_CHMODE,IRP$B_XQ_FUNC(R3) ; Set function request
        02D7   31    09D9  2821 190$:   BRW     QUEPKT                       ; Queue packet to driver
```

```
                        09DC  2823 ;+
                        09DC  2824 ; Take quota subroutine
                        09DC  2825 ;
                        09DC  2826 ;       Calculate buffer quota and check against user's quota
                        09DC  2827 ;
                        09DC  2828 ; Inputs:       R7 = Scratch
                        09DC  2829 ;               R6 = Scratch
                        09DC  2830 ;               R5 = UCB address
                        09DC  2831 ;               R4 = PCB address
                        09DC  2832 ;
                        09DC  2833 ;
                        09DC  2834 ; Outputs:      R7 = Quota taken
                        09DC  2835 ;               R6 = JIB address
                        09DC  2836 ;               R5 = UCB address
                        09DC  2837 ;               R4 = PCB address
                        09DC  2838 ;               R0 = Status
                        09DC  2839 ;
                        09DC  2840 ;       R1,R2 are destroyed.
                        09DC  2841 ;
                        09DC  2842 ; Implicit outputs:
                        09DC  2843 ;
                        09DC  2844 ;       BR to ABORTIO if quota is exceeded.
                        09DC  2845 ;
                        09DC  2846 ;-
                        09DC  2847 TAKE_QUOTA:
            50    D4    09DC  2848          CLRL    R0                          ; Assume failure
       00DA C5    B6    09DE  2849          INCW    UCB$W_XQ_BSZ(R5)            ; Round buffer size to even value
00DA C5    01    AA    09E2  2850          BICW    #1,UCB$W_XQ_BSZ(R5)         ;       ...
51     00D9 C5    9A    09E7  2851          MOVZBL  UCB$B_XQ_BFN(R5),R1          ; Get number of receive buffers
  52     42 A5    3C    09EC  2852          MOVZWL  UCB$W_DEVBUFSIZ(R5),R2       ; Get buffer size
       51   52    C4    09F0  2853          MULL    R2,R1                       ; Get needed quota
       57   51    3C    09F3  2854          MOVZWL  R1,R7                       ; Copy quota
       57   51    D1    09F6  2855          CMPL    R1,R7                       ; Overflow?
            30    12    09F9  2856          BNEQ    110$                        ; Br if yes - error
    7E    24 A4    DD    09FB  2857          MOVL    PCB$L_STS(R4),-(SP)         ; Save current PCB status
                         09FF  2858          SETBIT  #PCB$V_SSRWAIT,-            ; Do not go into a resource wait
                         09FF  2859                  PCB$L_STS(R4)               ;   just to check the quota
            53    DD    0A04  2860          PUSHL   R3                          ; Save R3
00000000'   GF    16    0A06  2861          JSB     G^EXE$BUFQUOPRC             ; Check quota
            53 BEDD     0A0C  2862          POPL    R3                          ; Restore R3
    24 A4    8E    DD    0A0F  2863          MOVL    (SP)+,PCB$L_STS(R4)         ; Restore previous PCB status
          02 50    E8    0A13  2864          BLBS    R0,50$                      ; Br if success
            10    11    0A16  2865          BRB     80$                         ; Else, return error
                         0A18  2866
56     0080 C4    D0    0A18  2867 50$:     MOVL    PCB$L_JIB(R4),R6            ; Get JIB address
  20 A6    57    C2    0A1D  2868          SUBL    R7,JIB$L_BYTCNT(R6)         ; Adjust quota
  24 A6    57    C2    0A21  2869          SUBL    R7,JIB$L_BYTLM(R6)          ; ..and byte limit
       50   01    9A    0A25  2870          MOVZBL  #1,R0                       ; Indicate success
            51    D4    0A28  2871 80$:     CLRL    R1                          ; No error return
            05    0A2A  2872 90$:     RSB
                         0A2B  2873
       50   14    9A    0A2B  2874 110$:    MOVZBL  #SS$_BADPARAM,R0            ; Setup error code
51     0451 8F    3C    0A2E  2875          MOVZWL  #NMA$C_PCLI_BFN,R1          ; Assume BAD BFN
            F5    11    0A33  2876          BPB     90$                         ; Return error
                         0A35  2877
                         0A35  2878 ;+
                         0A35  2879 ; CHECK_QUOTA - check SHARED unit's quota
```

```
                        0A35  2880 ;
                        0A35  2881 ; Inputs:       R9 = Scratch
                        0A35  2882 ;               R7 = Original UCB address
                        0A35  2883 ;               R5 = UCB address
                        0A35  2884 ;               R4 = PCB address
                        0A35  2885 ;
                        0A35  2886 ;
                        0A35  2887 ; Outputs:      R9 = Quota taken
                        0A35  2888 ;               R7 = Original UCB address
                        0A35  2889 ;               R5 = UCB address
                        0A35  2890 ;               R4 = PCB address
                        0A35  2891 ;               R0 = Status
                        0A35  2892 ;
                        0A35  2893 ;       R1,R2 are destroyed.
                        0A35  2894 ;
                        0A35  2895 ;-
                        0A35  2896 CHECK_QUOTA:
                        0A35  2897          PUSHQ   R6                          ; Save R6, R7
          55    DD      0A38  2898          PUSHL   R5                          ; Save UCB address
    55    57    D0      0A3A  2899          MOVL    R7,R5                       ; Copy original UCB address
          9D    10      0A3D  2900          BSBB    TAKE_QUOTA                  ; Charge quota to user
          55 8ED0       0A3F  2901          POPL    R5                          ; Restore UCB address
       16 50    E9      0A42  2902          BLBC    R0,90$                      ; Br if error
    59    57    D0      0A45  2903          MOVL    R7,R9                       ; Copy quota taken
51  0190 C5    3C      0A48  2904          MOVZWL  UCB$W_XQ_TOTQUO(R5),R1      ; Get total quota
    51    59    C0      0A4D  2905          ADDL    R9,R1                       ; Compute new total
    50    51    3C      0A50  2906          MOVZWL  R1,R0                       ; Copy quota
    50    51    D1      0A53  2907          CMPL    R1,R0                       ; Overflow?
          07    12      0A56  2908          BNEQ    110$                        ; Br if yes, error
    50    01    9A      0A58  2909          MOVZBL  #1,R0                       ; Else, return success
                        0A5B  2910 90$:     POPQ    R6                          ; Retore R6, R7
                05    CA5E  2911          RSB                                 ; Return to caller
                        0A5F  2912
20 A6    59    C0      0A5F  2913 110$:    ADDL    R9,JIB$L_BYTCNT(R6)         ; Restore quota
24 A6    59    C0      0A63  2914          ADDL    R9,JIB$L_BYTLM(R6)          ; ..and byte limit
   50    1C    3C      0A67  2915          MOVZWL  #SS$_EXQUOTA,R0             ; Return bad quota
         51    D4      0A6A  2916          CLRL    R1                          ; No parameter return
         ED    11      0A6C  2917          BRB     90$                         ; Exit with error
                        0A6E  2918
```

```
                        0A6E  2920          .SBTTL  SHR_UCB - CREATE SHARED UCB
                        0A6E  2921 ;++
                        0A6E  2922 ; SHR_UCB - CREATE SHARED UCB
                        0A6E  2923 ;
                        0A6E  2924 ; Functional description:
                        0A6E  2925 ;
                        0A6E  2926 ; This subroutine creates a shared UCB if this is the first SHARED user of the
                        0A6E  2927 ; particular protocol type.  Else, the already created SHARED UCB is found and
                        0A6E  2928 ; a shared data structure is added to the list of shared users of that protocol
                        0A6E  2929 ; type.
                        0A6E  2930 ;
                        0A6E  2931 ; Inputs:
                        0A6E  2932 ;
                        0A6E  2933 ;       R3 = IRP address
                        0A6E  2934 ;       R4 = PCB address
                        0A6E  2935 ;       R5 = UCB address
                        0A6E  2936 ;       R6 = CCB address
                        0A6E  2937 ;       R7 = Function code
                        0A6E  2938 ;       AP = Address of first function-dependent QIO parameter
                        0A6E  2939 ;
                        0A6E  2940 ;       IPL = FIPL
                        0A6E  2941 ;
                        0A6E  2942 ; Outputs:
                        0A6E  2943 ;
                        0A6E  2944 ;       R0 = Status return for request
                        0A6E  2945 ;       R1 = Bad parameter code (if bad parameter error code)
                        0A6E  2946 ;       R2,R7 are destroyed.
                        0A6E  2947 ;       All other registers are preserved.
                        0A6E  2948 ;
                        0A6E  2949 ;       IPL = ASTDEL
                        0A6E  2950 ;
                        0A6E  2951 ;--
                        0A6E  2952
                        0A6E  2953 SHR_UCB::                                    ; Setup shared protocol UCB
          0300 8F    BB 0A6E  2954          PUSHR   #^M<R8,R9>                  ; Save registers
 00000000'GF    16    0A72  2955          JSB     G^SCH$IOLOCKW               ; Lock I/O data base for write access
       57    55    D0 0A78  2956          MOVL    R5,R7                       ; Save UCB address
    50 21C4 8F    3C 0A7B  2957          MOVZWL  #SS$_DUPUNIT,R0             ; Assume 2 channels assigned to UCB
     5C A5    01    B1 0A80  2958          CMPW    #1,UCB$W_REFC(R5)           ; Is there only one reference to UCB?
              72    12 0A84  2959          BNEQ    23$                         ; Br if not, error (more than 1 channel)
    58 00D8 C5    9A 0A86  2960          MOVZBL  UCB$B_XQ_ACC(R5),R8         ; Save access mode
              03    E0 0A8B  2961          BBS     #UCB$V_XQ_SHARE,-           ; Br if we are already the SHARED UCB
        29 68 A5       0A8D  2962                  UCB$W_DEVSTS(R5),10$        ;
              54    DD 0A90  2963          PUSHL   R4                          ; Save PCB address
;RNGOOC1     54 00C8 C5    D0 0A92    .1  2965          MOVL    UCB$L_XQ_CDB(R5),R4         ; Get CDB address
-2          51 00CE C5    B0 0A97  2966          MOVW    UCB$W_XQ_PROTYP(R5),R1      ; Get protocol type
            1FFC    30 0A9C  2967          BSBW    MATCH_PROTYP                ; Try to match protocol type
              54 8ED0 0A9F  2968          POPL    R4                          ; Restore PCB address
           14 50    E8 0AA2  2969          BLBS    R0,10$                      ; Br if found
        55    57    D0 0AA5  2970          MOVL    R7,R5                       ; Else, get back old UCB address
     2C A5    D4 0AA8  2971          CLRL    UCB$L_PID(R5)               ; Make this UCB shareable
  50 1C A5    D0 0AAB  2972          MOVL    UCB$L_ORB(R5),R0            ; Get the ORB address
              60    D4 0AAF  2973          CLRL    ORB$L_OWNER(R0)             ;  clear owner UIC as well
        00C4 C5    D4 0AB1  2974          CLRL    UCB$L_XQ_DEFUSR(R5)         ; Clear default user
              08    A8 0AB5  2975          BISW    #UCB$M_XQ_SHARE,-           ; Indicate that UCB is in SHARED mode
           68 A5       0AB7  2976                  UCB$W_DEVSTS(R5)            ;
                       0AB9  2977 ;
```

```
                                0AB9  2978 ; Allocate a share data structure and link it in
                                0AB9  2979 ;
      58   02   91   0AB9  2980 10$:    CMPB    #NMA$C_ACC_LIM,R8       ; Is the new user a limited user?
                0B   13   0ABC  2981          BEQL    20$                     ; Br if yes - okay
50    0840 8F   3C   0ABE  2982          MOVZWL  #SS$_DEVALLOC,R0        ; Assume protocol already allocated
      00C4 C5   D5   0AC3  2983          TSTL    UCB$L_XQ_DEFUSR(R5)     ; Is there already a default user?
                2F   12   0AC7  2984          BNEQ    23$                     ; Br if YES - error
                00C8 30   0AC9  2985 20$:    BSBW    CHECK_PARAM             ; Check out all parameters
              2B 50  E9   0ACC  2986          BLBC    R0,25$                  ; Br on error
               FF63  30   0ACF  2987          BSBW    CHECK_QUOTA             ; Check our quota
              25 50  E9   0AD2  2988          BLBC    R0,25$                  ; Br if error
      51      2A   9A   0AD5  2989          MOVZBL  #SHR_C_LENGTH,R1        ; Get size of structure to allocate
              53   DD   0AD8  2990          PUSHL   R3                      ; Save IRP address
 00000000'GF  16   0ADA  2991          JSB     G^EXE$ALLOCBUF          ; Allocate buffer, reset IPL to ASTDEL
              53 8ED0   0AE0  2992          POPL    R3                      ; Restore IRP address
            1A 50  E8   0AE3  2993          BLBS    R0,30$                  ; Br if success
50    0124 8F   3C   0AE6  2994          MOVZWL  #SS$_INSFMEM,R0         ; Else, return error reason
51    0080 C4   D0   0AEB  2995          MOVL    PCB$L_JIB(R4),R1        ; Get JIB address
  20 A1   59   C0   0AF0  2996          ADDL    R9,JIB$L_BYTCNT(R1)     ; Restore quota
  24 A1   59   C0   0AF4  2997          ADDL    R9,JIB$L_BYTLM(R1)      ; ..and byte limit
      51   D4   0AF8  2998 23$:    CLRL    R1                      ; No bad parameter code
      55   57   D0   0AFA  2999 25$:    MOVL    R7,R5                   ; Get back the OLD UCB address
         0085 31   0AFD  3000          BRW     80$                     ; Exit with error
                0B00  3001 ;
                0B00  3002 ; Initialize shared (SHR) data structure
                0B00  3003 ;
                0B00  3004 30$:    ASSUME  SHR_L_QFL EQ 0
                0B00  3005          ASSUME  SHR_L_QBL EQ SHR_L_QFL+4
           82   7C   0B00  3006          CLRQ    (R2)+                   ; Zero LINK pointers
                0B02  3007          ASSUME  SHR_W_SIZE EQ SHR_L_QBL+4
      82   51   BC   0B02  3008          MOVW    R1,(R2)+                ; Save size
                0B05  3009          ASSUME  SHR_B_TYPE EQ SHR_W_SIZE+2
           52   D6   0B05  3010          INCL    R2                      ; Filled by EXE$ALLOCBUF routine
                0B07  3011          ASSUME  SHR_B_STS EQ SHR_B_TYPE+1
      82   01   90   0B07  3012          MOVB    #SHR_STS_M_INITED,(R2)+ ; Initialize SHR status
                0B0A  3013          ASSUME  SHR_L_PID EQ SHR_B_STS+1
  82    0C A3   D0   0B0A  3014          MOVL    IRP$L_PID(R3),(R2)+     ; Save users PID and CHAN for
                0B0E  3015          ASSUME  SHR_W_CHAN EQ SHR_L_PID+4
  82    28 A3   B0   0B0E  3016          MOVW    IRP$W_CHAN(R3),(R2)+    ;  for future lookups
                0B12  3017          ASSUME  SHR_G_DEST EQ SHR_W_CHAN+2
82    00D0 C7   D0   0B12  3018          MOVL    UCB$G_XQ_DES(R7),(R2)+  ; Save destination address
82    00D4 C7   B0   0B17  3019          MOVW    UCB$G_XQ_DES+4(R7),(R2)+ ;
                0B1C  3020          ASSUME  SHR_Q_QUEUES EQ SHR_G_DEST+6
      51   02   9A   0B1C  3021          MOVZBL  #SHR_C_QUEUES,R1        ; Get number of queues in structure
      82   62   DE   0B1F  3022 40$:    MOVAL   (R2),(R2)+              ; Set forward link pointer
  82    FC A2   D0   0B22  3023          MOVL    -4(R2),(R2)+            ; Set backward link pointer
        F6 51   F5   0B26  3024          SOBGTR  R1,40$                  ; Loop if more listheads
                0B29  3025          ASSUME  SHR_W_QUOTA EQ SHR_Q_QUEUES+<8*SHR_C_QUEUES>
      82   59   B0   0B29  3026          MOVW    R9,(R2)+                ; Initialize quota
                0B2C  3027 ;
                0B2C  3028          ASSUME  UCB$V_XQ_INITED EQ 0
    0C 68 A5   E9   0B2C  3029          BLBC    UCB$W_DEVSTS(R5),50$    ; Br if UCB not initialized
00CC C5   59   A0   0B30  3030          ADDW    R9,UCB$W_XQ_QUOTA(R5)   ; Add to the current quota
0190 C5   59   A0   0B35  3031          ADDW    R9,UCB$W_XQ_TOTQUO(R5)  ;  and the total quota
            0A   11   0B3A  3032          BRB     55$                     ; Continue
00CC C5   59   B0   0B3C  3033 50$:    MOVW    R9,UCB$W_XQ_QUOTA(R5)   ; Set current quota
0190 C5   59   B0   0B41  3034          MOVW    R9,UCB$W_XQ_TOTQUO(R5)  ;  and total quota
```

```
                52   2A   C2  0B46  3035 55$:    SUBL    #SHR_C_LENGTH,R2          ; Backup to beginning of structure
                58   02   91  0B49  3036         CMPB    #NMA$C_ACC_LIM,R8         ; Is this for limited use?
                     07   13  0B4C  3037         BEQL    60$                       ; Br if YES
          00C4  C5   52   D0  0B4E  3038         MOVL    R2,UCB$L_XQ_DEFUSR(R5)    ; Else, save default user address
                     05   11  0B53  3039         BRB     65$                       ; Skip linking onto list
          009C  D5   62   0E  0B55  3040 60$:    INSQUE  (R2),@UCB$Q_XQ_SHARE+4(R5) ; Link user into shared user list
                   5C   A5   B6  0B5A  3041 65$:    INCW    UCB$W_REFC(R5)            ; Increment the ref count on the
                                0B5D  3042                                           ;  UCB to be used
                57   55   D1  0B5D  3043         CMPL    R5,R7                     ; Was this the original UCB?
                     20   13  0B60  3044         BEQL    70$                       ; Br if YES - no more work to do
                   5C   A5   B6  0B62  3045         INCW    UCB$W_REFC(R5)            ; Else, increment REFC (for $DASSGN)
                1C A3   55   D0  0B65  3046         MOVL    R5,IRP$L_UCB(R3)          ; Return new UCB address
                66   55   D0  0B69  3047         MOVL    R5,CCB$L_UCB(R6)          ;  in CCB also.
                     28   BB  0B6C  3048         PUSHR   #^M<R3,R5>                ; Save IRP, real UCB address
                55   57   D0  0B6E  3049         MOVL    R7,R5                     ; Copy old UCB address
                   5C   A5   B7  0B71  3050         DECW    UCB$W_REFC(R5)            ; Decrement the reference count
          00000000'GF   16  0B74  3051         JSB     G^IOC$CREDIT_UCB          ; Restore UCB quota to JIB
          00000000'GF   16  0B7A  3052         JSB     G^IOC$DELETE_UCB          ; Delete the old UCB
                     28   BA  0B80  3053         POPR    #^M<R3,R5>                ; Restore IRP, UCB address
                50   01   9A  0B82  3054 70$:    MOVZBL  S^#SS$_NORMAL,R0          ; Return success
                0300 8F   BA  0B85  3055 80$:    POPR    #^M<R8,R9>                ; Restore registers, R4 is PCB address
                     0B   BB  0B89  3056         PUSHR   #^M<R0,R1,R3>             ; Save IRP address, status return
          00000000'GF   16  0B8B  3057         JSB     G^SCH$IOUNLOCK            ; Unlock I/O data base
                     0B   BA  0B91  3058         POPR    #^M<R0,R1,R3>             ; Restore IRP address, status return
                          05  0B93  3059         RSB                               ; Return to caller
```

```
                     0B94  3061          .SBTTL  CHECK_PARAM - CHECK SHARED USERS PARAMETERS
                     0B94  3062 ;++
                     0B94  3063 ; CHECK_PARAM - CHECK SHARED USERS PARAMETERS
                     0B94  3064 ;
                     0B94  3065 ; Functional description:
                     0B94  3066 ;
                     0B94  3067 ; Validate all parameters between the requesting SHARED user and the old
                     0B94  3068 ; existing SHARED user to make sure that are the same.
                     0B94  3069 ;
                     0B94  3070 ; Inputs:
                     0B94  3071 ;       R5 = UCB address of existing shared user
                     0B94  3072 ;       R7 = UCB address of new shared user
                     0B94  3073 ;       R8 = Protocol access mode
                     0B94  3074 ;
                     0B94  3075 ; Outputs:
                     0B94  3076 ;       R0 = Status of request
                     0B94  3077 ;       R1 = Bad parameter code if validation failed
                     0B94  3078 ;
                     0B94  3079 ;--
                     0B94  3080
                     0B94  3081 CHECK_PARAM:                                 ; Check user parameters
          56    DD   0B94  3082          PUSHL   R6                          ; Save registers
    50    00DA C5   9E 0B96  3083          MOVAB   UCB$B_XQ_SHRPRM(R5),R0      ; Get address of current parameters
    56    00DA C7   9E 0B9B  3084          MOVAB   UCB$B_XQ_SHRPRM(R7),R6      ; Get address of new parameters
          52    08   9A 0BA0  3085          MOVZBL  #UCB$C_XQ_SHRPRM,R2         ; Set size of parameter list
                     0BA3  3086 ;
                     0BA3  3087 ; Validate all user settable parameters except for Physical address
                     0BA3  3088 ;
          66    60   91 0BA3  3089 10$:     CMPB    (R0),(R6)                   ; Match?
                28   12 0BA6  3090          BNEQ    70$                         ; Br if no
          86    80   90 0BA8  3091          MOVB    (R0)+,(R6)+                 ; Store current value in UCB
            F5  52   F5 0BAB  3092          SOBGTR  R2,10$                      ; Loop if more to check
                     0BAE  3093 ;
                     0BAE  3094 ; NOW, check if user has given a hardware physical address.
                     0BAE  3095 ;
                     0BAE  3096 ;       R0 = Address of parameters in UCB
                     0BAE  3097 ;       R6 = Address of parameters in CDB
                     0BAE  3098 ;
                     0BAE  3099          ASSUME  CDB_G_PHA EQ CDB_B_CON+1
                     0BAE  3100          ASSUME  UCB$G_XQ_PHA EQ UCB$B_XQ_CON+1
FFFFFFFF  8F   D1    0BAE  3101          CMPL    #-1,(R6)                    ; Is user physical address defined?
          66                0BB4
          08   12    0BB5  3102          BNEQ    20$                         ; Br if yes
      FFFF  8F   B1 0BB7  3103          CMPW    #-1,4(R6)                   ; Is user physical address defined?
        04  A6                0BBB
          0F   13    0BBD  3104          BEQL    30$                         ; Br if not
51    0B04  8F   3C 0BBF  3105 20$:     MOVZWL  #NMA$C_PCLI_PHA,R1          ; Assume bad physical address
      86    80   D1 0BC4  3106          CMPL    (R0)+,(R6)+                 ; Physical address match??
          10   12    0BC7  3107          BNEQ    80$                         ; Br if no
      86    80   B1 0BC9  3108          CMPW    (R0)+,(R6)+                 ; Still match??
          0B   12    0BCC  3109          BNEQ    80$                         ; Br if no
                     0BCE  3110 ;
                     0BCE  3111 ; If this is the shared default user, then set the multicast address list.
                     0BCE  3112 ;
        000F   30    0BCE  3113 30$:     BSBW    SET_MULTIN                  ; Set new multicast address list
          09   11    0BD1  3114          BRB     100$                        ; Exit
                     0BD3  3115 ;
```

```
                                 0BD3  3116 ; Error on parameter validation
                                 0BD3  3117 ;
51    F5CC CF42    3C     0BD3  3118 70$:    MOVZWL  BAD_PARAM_TBL-2[R2],R1  ; Return parameter code
      50    14     9A     0BD9  3119 80$:    MOVZBL  S^#SS$_BADPARAM,R0      ; Return bad parameter error
            56 8ED0     0BDC  3120 100$:    POPL    R6                      ; Restore registers
                  05     0BDF  3121          RSB
```

```
                     0BE0  3123          .SBTTL  SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
                     0BE0  3124 ;++
                     0BE0  3125 ; SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
                     0BE0  3126 ;
                     0BE0  3127 ; Functional description:
                     0BE0  3128 ;
                     0BE0  3129 ; Copy the multicast address list from the old UCB to the new UCB.
                     0BE0  3130 ; This operation is only done for the SHARED DEFAULT user of the PROTOCOL.
                     0BE0  3131 ;
                     0BE0  3132 ; Inputs:
                     0BE0  3133 ;       R5 = UCB address of existing shared user
                     0BE0  3134 ;       R7 = UCB address of new shared user
                     0BE0  3135 ;       R8 = Protocol access mode
                     0BE0  3136 ;
                     0BE0  3137 ; Outputs:
                     0BE0  3138 ;       R0 = Always success
                     0BE0  3139 ;       All other are preserved.
                     0BE0  3140 ;
                     0BE0  3141 ;--
                     0BE0  3142
                     0BE0  3143 SET_MULTIN:                                  ; Set new multicast list
58   01   91   0BE0  3144          CMPB    #NMA$C_ACC_SHR,R8                 ; Is this the shared default user?
     15   12   0BE3  3145          BNEQ    50$                               ; Br if not
     3E   BB   0BE5  3146          PUSHR   #^M<R1,R2,R3,R4,R5>               ; Save registers
0048 8F   28   0BE7  3147          MOVC3   #6*MAX_C_MLT,UCB$G_XQ_MULTI(R7),- ; Copy multicast list
00EC C7
00EC C5        0BEB
               0BEE
               0BF1  3148                  UCB$G_XQ_MULTI(R5)                ;
     3E   BA   0BF1  3149          POPR    #^M<R1,R2,R3,R4,R5>               ; Restore registers
00E9 C7   90   0BF3  3150          MOVB    UCB$B_XQ_MULTI(R7),-              ; Copy the number of valid addresses
00E9 C5        0BF7  3151                  UCB$B_XQ_MULTI(R5)                ;
50   01   9A   0BFA  3152 50$:     MOVZBL  #SS$_NORMAL,R0                    ; Return success
          05   0BFD  3153          RSB                                       ; Return to caller
```

```
                        0BFE  3155          .SBTTL  SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
                        0BFE  3156 ;++
                        0BFE  3157 ; SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
                        0BFE  3158 ;
                        0BFE  3159 ; Functional description:
                        0BFE  3160 ;
                        0BFE  3161 ; Process read status and read counters requests.
                        0BFE  3162 ;
                        0BFE  3163 ; The QIO parameters for SENSEMODE are:
                        0BFE  3164 ;
                        0BFE  3165 ;       P1 = Optional address of quadword buffer
                        0BFE  3166 ;       P2 = Optional address of buffer descriptor for extended characteristics
                        0BFE  3167 ;
                        0BFE  3168 ; The SUBFUNCTION modifiers are as follows:
                        0BFE  3169 ;
                        0BFE  3170 ; 1)    READ PARAMETERS -- NO MODIFIER.
                        0BFE  3171 ;       This function reads the QNA parameters and returns them to the user.
                        0BFE  3172 ;
                        0BFE  3173 ; 2)    READ COUNTERS -- IO$M_RD_COUNT SET.
                        0BFE  3174 ;       This function reads the QNA counters and returns them to the user.
                        0BFE  3175 ;
                        0BFE  3176 ;       CLEAR COUNTERS -- IO$M_CLR_COUNT SET.
                        0BFE  3177 ;       This modifier must be used with the read counters modifier to clear
                        0BFE  3178 ;       the counters as they are read.
                        0BFE  3179 ;
                        0BFE  3180 ;
                        0BFF  3181 ; Inputs:
                        0BFE  3182 ;
                        0BFE  3183 ;       R3 = IRP address
                        0BFE  3184 ;       R4 = PCB address
                        0BFE  3185 ;       R5 = UCB address
                        0BFE  3186 ;       R6 = CCB address
                        0BFE  3187 ;       R7 = Function code
                        0BFE  3188 ;       AP = Address of first function-dependent QIO parameter
                        0BFE  3189 ;
                        0BFE  3190 ;       IPL = ASTDEL
                        0BFE  3191 ;
                        0BFE  3192 ; Outputs:
                        0BFE  3193 ;
                        0BFE  3194 ;       R0 = Status return of SENSEMODE request
                        0BFE  3195 ;
                        0BFE  3196 ;       R1,R2,R6,R7 are destroyed.
                        0BFE  3197 ;
                        0BFE  3198 ;--
                        0BFE  3199
                        0BFE  3200 SENSEMODE_FDT::                              ; SENSE MODE I/O FDT processing
50   0084 8F    3C      0BFE  3201          MOVZWL  #SS$_DEVOFFLINE,R0          ; Assume unit if offline
           04    E1      0C03  3202          BBC     #UCB$V_ONLINE,-             ; Br if unit not online
     23 64 A5            0C05  3203                  UCB$W_STS(R5),15$           ;
57   20 A3    B0         0C08  3204          MOVW    IRP$W_FUNC(R3),R7           ; Get entire function code
14 57   09    E0         0C0C  3205          BBS     #IO$V_CTRL,R7,10$           ; Br if line request
                        0C10  3206 ;
                        0C10  3207 ; Check if read circuit counters
                        0C10  3208 ;
03 57   08    E1         0C10  3209          BBC     #IO$V_RD_COUNT,R7,5$        ; Br if not read circuit counters
       00D2   31         0C14  3210          BRW     READ_CIRC_CTR               ; Else, get the circuit counters
   50   01    9A         0C17  3211 5$:      MOVZBL  S^#SS$_NORMAL,R0            ; Return success
```

```
                 51    44 A5    D0  0C1A  3212          MOVL    UCB$L_DEVDEPEND(R5),R1  ; Get device dependent information
            00000000'GF    17  0C1E  3213          JMP     G^EXE$FINISHIC          ; Complete the I/O request
                                0C24  3214
                  06 57    08  E1  0C24  3215 10$:     BBC     #IO$V_RD_COUNT,R7,20$   ; Br if not read counters
                                0C28  3216 ;
                                0C28  3217 ; Read counters - modifier RD_COUNT
                                0C28  3218 ;
                       009F     31  0C28  3219          BRW     READ_LINE_CTR           ; Get the line counters
                                0C2B  3220
                       '712     31  0C2B  3221 15$:     BRW     ABORTIO                 ; Abort the I/O request
                                0C2E  3222 ;
                                0C2E  3223 ; Read parameters - no modifier
                                0C2E  3224 ;
                    44 A3    B4  0C2E  3225 20$:     CLRW    IRP$W_XQ_P2SIZ(R3)      ; No return data
;RNG0001          54   00C8 C5  D0  0C31    .1          MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
;RNG0001              51   08  3C  0C36    .2          MOVZWL  #8,R1                   ; Size of P1 buffer if present
-3                      017E  30  0C39  3229          BSBW    CHECK_BUFS              ; Check P1 and P2 buffers
                     50   01  9A  0C3C  3230          MOVZBL  S^#SS$_NORMAL,R0        ; Assume success
                  3A A3   50  B0  0C3F  3231          MOVW    R0,IRP$W_XQ_STATUS(R3)  ;       ...
                  38 A3   51  B0  0C43  3232          MOVW    R1,IRP$W_XQ_USERSIZ(R3) ; Save user P2 buffer length
                          47  13  0C47  3233          BEQL    40$                     ; Br if no P2 buffer present
                                0C49  3234
                  40 A3   52  D0  0C49  3235          MOVL    R2,IRP$L_XQ_P2BUF(R3)   ; Save user P2 buffer address
                        1B21  30  0C4D  3236          BSBW    RETURN_P2               ; Return the P2 parameters
                  44 A3   50  B0  0C50  3237          MOVW    R0,IRP$W_XQ_P2SIZ(R3)   ; Set size of return data
                          54  D5  0C54  3238          TSTL    R4                      ; Is CDB present?
                          2C  13  0C56  3239          BEQL    30$                     ; Br if no - okay to return now
;RNG0001          51   50   0A  C1  0C58    .1          ADDL3   #10,R0,R1               ; Check if default physical
-1                                      0C5C  3241                                  ;  address can fit
                  38 A3   51  B1  0C5C  3242          CMPW    R1,IRP$W_XQ_USERSIZ(R3)         ; Is buffer big enough for
                                0C60  3243                                  ; default physical address?
                          1C  1A  0C60  3244          BGTRU   25$                     ; Br if no
                  44 A3   0A  A0  0C62  3245          ADDW    #10,IRP$W_XQ_P2SIZ(R3)  ; Add DPA to return buffer size
                  40 A3   50  C1  0C66  3246          ADDL3   R0,IRP$L_XQ_P2BUF(R3),R2        ; Get buffer address
                          52      0C6A
                                0C6B  3247                                  ; past end of return data
             00061438     8F  D0  0C6B  3248          MOVL    #<6@16>+NMA$C_PCLI_HWA!-; Store parameter code + size
                          82      0C71
                                0C72  3249                  PRM_TYP_M_STRING,(R2)+  ;  and indicate this is a string
            82   0250 C4  D0  0C72  3250          MOVL    CDB_G_HWA(R4),(R2)+     ; Store Default Physical Address
            82   0254 C4  B0  0C77  3251          MOVW    CDB_G_HWA+4(R4),(R2)+   ;
                          06  11  0C7C  3252          BRB     30$                     ; All is okay
                 0601 8F  B0  0C7E  3253 25$:     MOVW    #SS$_BUFFEROVF,IRP$W_XQ_STATUS(R3) ; Return partial success
                    3A A3      0C82
                  50   44 A3   3C  0C84  3254 30$:     MOVZWL  IRP$W_XQ_P2SIZ(R3),R0   ; Get size of user return data
            50    50    10  78  0C88  3255          ASHL    #16,R0,R0               ; Shift size of buffer return
                  50   3A A3   B0  0C8C  3256          MOVW    IRP$W_XQ_STATUS(R3),R0  ; Get status
                                0C90  3257
                          54  D5  0C90  3258 40$:     TSTL    R4                      ; Is there a CDB?
                          15  13  0C92  3259          BEQL    50$                     ; Br if no CDB yet!
                  52   3C A3   D0  0C94  3260          MOVL    IRP$L_XQ_USERBUF(R3),R2 ; Retrieve P1 buffer address
                          0F  13  0C98  3261          BEQL    50$                     ; Br if none
                  62   40 A5   7D  0C9A  3262          MOVQ    UCB$B_DEVCLASS(R5),(R2) ; Else, return characteristics
                 010C C4      C8  0C9E  3263          BISL    CDB_L_DEVDEPEND(R4),4(R2) ;     ...
                    04 A2      0CA2
            51   010C C4      D0  0CA4  3264          MOVL    CDB_L_DEVDEPEND(R4),R1  ; Get device dependent info
                  51   44 A5   C8  0CA9  3265 50$:     BISL    UCB$L_DEVDEPEND(R5),R1  ; ..from UCB also
```

```
                00000000'GF   17  0CAD  3266          JMP     G^EXE$FINISHIO          ; Complete the I/O request
                              0CB3  3267
                              0CB3  3268 ;
                              0CB3  3269 ; Queue I/O request to driver
                              0CB3  3270 ;
                              0CB3  3271 QUEPKT:                                    ; Queue packet to driver
                              0CB3  3272          SETIPL  UCB$B_FIPL(R5)            ; Raise IPL to fork IPL
                00000000'GF   16  0CB7  3273          JSB     G^IOC$INITIATE          ; Intiate the I/O request
                00000000'GF   17  0CBD  3274          JMP     G^EXE$QIORETURN         ; Lower IPL, and RET
```

```
                     OCC3  3276          .SBTTL  READ_LINE_CTR - READ THE LINE COUNTERS
                     OCC3  3277          .SBTTL  READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
                     OCC3  3278 ;++
                     OCC3  3279 ; READ_LINE_CTR - READ THE LINE COUNTERS
                     OCC3  3280 ; READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
                     OCC3  3281 ;
                     OCC3  3282 ; Functional description:
                     OCC3  3283 ;
                     OCC3  3284 ; Process read circuit counters request.
                     OCC3  3285 ;
                     OCC3  3286 ; The QIO parameters for SENSEMODE are:
                     OCC3  3287 ;
                     OCC3  3288 ;       P2 = Address of buffer descriptor for counters
                     OCC3  3289 ;
                     OCC3  3290 ;
                     OCC3  3291 ; Inputs:
                     OCC3  3292 ;
                     OCC3  3293 ;       R3 = IRP address
                     OCC3  3294 ;       R4 = PCB address
                     OCC3  3295 ;       R5 = UCB address
                     OCC3  3296 ;       R6 = CCB address
                     OCC3  3297 ;       R7 = Function code and modifier bits
                     OCC3  3298 ;       AP = Address of first function-dependent QIO parameter
                     OCC3  3299 ;
                     OCC3  3300 ; Outputs:
                     OCC3  3301 ;
                     OCC3  3302 ;       R0 = Status return of SENSEMODE request
                     OCC3  3303 ;
                     OCC3  3304 ;       R1,R2,R6,R7 are destroyed.
                     OCC3  3305 ;
                     OCC3  3306 ;--
                     OCC3  3307          .ENABL  LSB
                     OCC3  3308 ABORT_IRP:
           0700 8F    BA   OCC3  3309          POPR    #^M<R8,R9,R10>          ; Restore  registers
              F676    31   OCC7  3310          BRW     ABORTIO                 ; Abort the I/O request
                           OCCA  3311
                           OCCA  3312 READ_LINE_CTR:                          ; Read the line counters
           0700 8F    BB   OCCA  3313          PUSHR   #^M<R8,R9,R10>          ; Save registers
        58 F437 CF    9E   OCCE  3314          MOVAB   LINE_CTR,R8             ; Get address of counter format table
           59   14    3C   OCD3  3315          MOVZWL  #LINE_CTR_SIZE,R9       ; Get number of entries in table
        56 006A 8F    3C   OCD6  3316          MOVZWL  #LINE_CTR_BUFSIZ,R6     ; Get size of system P2 buffer
;RNG0001 50 0124 8F    3C   OCDB    .1         MOVZWL  #SS$_INSFMEM,R0         ; Assume no CDB
;RNG0001 5A 00C8 C5    D0   OCE0    .2         MOVL    UCB$L_XQ_CDB(R5),R10    ; Get CDB address
-3                    DC   13   OCE5  3320          BEQL    ABORT_IRP       ; Br if none, abort the I/O
                      12   11   OCE7  3321          BRB     10$             ; Else, Continue in common code
                           OCE9  3322
                           OCE9  3323 READ_CIRC_CTR:                          ; Read the circuit counters
           0700 8F    BB   OCE9  3324          PUSHR   #^M<R8,R9,R10>          ; Save registers
        58 F468 CF    9E   OCED  3325          MOVAB   CIRC_CTR,R8             ; Get address of counter format table
           59   06    3C   OCF2  3326          MOVZWL  #CIRC_CTR_SIZE,R9       ; Get number of entries in table
           56   20    3C   OCF5  3327          MOVZWL  #CIRC_CTR_BUFSIZ,R6     ; Get size of system P2 buffer
           5A   55    D0   OCF8  3328          MOVL    R5,R10                  ; Use UCB for counters
                           OCFB  3329
              00BE    30   OCFB  3330 10$:     BSBW    CHECK_P2                ; Check the P2 buffer
           50   14    9A   OCFE  3331          MOVZBL  S^#SS$_BADPARAM,R0      ; Assume zero length buffer
     38 A3   51    B0   0D01  3332          MOVW    R1,IRP$W_XQ_USERSIZ(R3) ; Save size of user P2 buffer
                      BC   13   0D05  3333          BEQL    ABORT_IRP       ; Br if no buffer
```

```
                51    56    D0   0D07  3354          MOVL    R6,R1                   ; Get size of system P2 buffer
                   00E9    30   0D0A  3355          BSBW    ALLOC_P2BUF             ; Allocate the buffer
                  B3 50    E9   0D0D  3356          BLBC    R0,ABORT_IRP            ; Br if error
              51    2C A3    D0   0D10  3357          MOVL    IRP$L_SVAPTE(R3),R1     ; Get system P2 buffer address
              04 A1    52    D0   0D14  3358          MOVL    R2,P2B_L_BUFFER(R1)     ; Save user P2 buffer address
                52    61    D0   0D18  3359          MOVL    P2B_L_POINTER(R1),R2    ; Get address of data portion of buffer
                                 0D1B  3360 ;
                                 0D1B  3361 ; Get the counters kept by the driver
                                 0D1B  3362 ;
                50    88    3C   0D1B  3363 20$:     MOVZWL  (R8)+,R0                ; Get counter code
                82    50    B0   0D1E  3364          MOVW    R0,(R2)+                ; Return counter type code
                51    88    3C   0D21  3365          MOVZWL  (R8)+,R1                ; Get offset word
                51    5A    C0   0D24  3366          ADDL    R10,R1                  ; Point to counter in UCB
           50   03    0C    EF   0D27  3367          EXTZV   #NMA$V_CNT_MAP,#3,R0,R0 ; Get width + bit map indicator
                      50         0D2B
                                 0D2C  3368          CASE    R0,TYPE=B,LIMIT=#2,<-   ; Dispatch on width and bit map
                                 0D2C  3369                  30$,-                   ; 8 bit counter
                                 0D2C  3370                  30$,-                   ; 8 bit counter + bit map
                                 0D2C  3371                  40$,-                   ; 16 bit counter
                                 0D2C  3372                  35$,-                   ; 16 bit counter + bit map
                                 0D2C  3373                  35$>                    ; 32 bit counter
                                 0D3A  3374
                                 0D3A  3375 30$:     BUG_CHECK NOBUFPCKT,FATAL
                                 0D3E  3376
                                 0D3E  3377 ;
                                 0D3E  3378 ; 32 BIT counter/ 16 BIT counter + bitmap
                                 0D3E  3379 ;
                82    81    B0   0D3E  3380 35$:     MOVW    (R1)+,(R2)+             ; Store counter in buffer
              03 57    0A    E1   0D41  3381          BBC     #IO$V_CLR_COUNT,R7,40$  ; Br if not clear counter operation
                   FE A1    B4   0D45  3382          CLRW    -2(R1)                  ; Else, clear the counter as well
                                 0D48  3383 ;
                                 0D48  3384 ; 16 BIT counter
                                 0D48  3385 ;
                82    61    B0   0D48  3386 40$:     MOVW    (R1),(R2)+              ; Store counter in buffer
              02 57    0A    E1   0D4B  3387          BBC     #IO$V_CLR_COUNT,R7,50$  ; Br if not clear counter operation
                61    B4   0D4F  3388          CLRW    (R1)                    ; Else, clear the counter as well
                   C7 59    F5   0D51  3389 50$:     SOBGTR  R9,20$                  ; Loop if more
                51    01    9A   0D54  3390          MOVZBL  S^#SS$_NORMAL,R1        ; Assume success
                   38 A3    B1   0D57  3391          CMPW    IRP$W_XQ_USERSIZ(R3),-  ; Is user's buffer big enough?
                   32 A3         0D5A  3392                  IRP$W_BCNT(R3)          ;
                      0A    1E   0D5C  3393          BGEQU   60$                     ; Br if yes
                   38 A3    B0   0D5E  3394          MOVW    IRP$W_XQ_USERSIZ(R3),-  ; Else, set size to minimum
                   32 A3         0D61  3395                  IRP$W_BCNT(R3)          ; Of both
           51   0601 8F    B0   0D63  3396          MOVW    #SS$_BUFFEROVF,R1       ; Set partial success
             50    30 A3    D0   0D68  3397 60$:     MOVL    IRP$Q_BCNT-2(R3),R0     ; Get size of buffer returned in
                                 0D6C  3398                                          ; ..high word of R0
                50    51    B0   0D6C  3399          MOVW    R1,R0                   ; Get status return
              51    44 A5    D0   0D6F  3400          MOVL    UCB$L_DEVDEPEND(R5),R1  ; Get device dependent info
                 0700 8F    BA   0D73  3401          POPR    #^M<R8,R9,R10>          ; Restore registers
         00000000'GF    17   0D77  3402          JMP     G^EXE$FINISHIO          ; Complete the I/O request
                                 0D7D  3403
                                 0D7D  3404          .DSABL  LSB
                                 0D7D  3405
```

```
                     0D7D  3387          .SBTTL  GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
                     0D7D  3388 ;++
                     0D7D  3389 ; GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
                     0D7D  3390 ;
                     0D7D  3391 ; Functional description:
                     0D7D  3392 ;
                     0D7D  3393 ; This routine saves the P2 buffer for later use by the driver.
                     0D7D  3394 ; The P2 buffer is saved by allocating the appropriate amount of memory from
                     0D7D  3395 ; non-paged pool.   The user's quota is checked before the allocation is made.
                     0D7D  3396 ; And the non-paged pool buffer is charged against the user's quota. The P2
                     0D7D  3397 ; system buffer address is passed in IRP$L_SVAPTE of the IRP.
                     0D7D  3398 ;
                     0D7D  3399 ;
                     0D7D  3400 ; Inputs:
                     0D7D  3401 ;
                     0D7D  3402 ;       R3 = IRP address
                     0D7D  3403 ;       R4 = PCB address
                     0D7D  3404 ;       R5 = UCB address
                     0D7D  3405 ;
                     0D7D  3406 ; Outputs:
                     0D7D  3407 ;
                     0D7D  3408 ;       R0 = status of buffers
                     0D7D  3409 ;
                     0D7D  3410 ;       R3-R5 are preserved.
                     0D7D  3411 ;
                     0D7D  3412 ;--
                     0D7D  3413
                     0D7D  3414 GET_CHAR_BUF:                                    ; Get characteristics buffer
                     0D7D  3415 ;
                     0D7D  3416 ; Check access to P2 buffer and check process's buffer quota
                     0D7D  3417 ;
 51    04 AC    D0   0D7D  3418 10$:    MOVL    P2(AP),R1                      ; Get address P2 char buf desc
          33    13   0D81  3419         BEQL    40$                            ; Br if no P2 buffer
          53    DD   0D83  3420         PUSHL   R3                             ; Save R3
00000000'GF    16   0D85  3421         JSB     G^EXE$PROBER_DSC               ; Check access to buffer
       0E 50   E9   0D88  3422         BLBC    R0,15$                         ; Br if error
    51    51    3C   0D8E  3423         MOVZWL  R1,R1                          ; Get the length as a word
          52    DD   0D91  3424         PUSHL   R2                             ; Save R2
00000000'GF    16   0D93  3425         JSB     G^EXE$BUFQUOPRC                ; Check for buffered quota
          52 8ED0   0D99  3426         POPL    R2                             ; Restore R2
          53 8ED0   0D9C  3427 15$:    POPL    R3                             ; Restore R3
       01 50    E8   0D9F  3428         BLBS    R0,30$                         ; Branch if quota ok
                05   0DA2  3429 20$:    RSB                                    ; Return
                     0DA3  3430
                     0DA3  3431 ;
                     0DA3  3432 ; Quota OKAY, allocate buffer and copy info.
                     0DA3  3433 ;
      0050   30      0DA3  3434 30$:    BSBW    ALLOC_P2BUF                    ; Allocate buffer
     10 50   E9      0DA6  3435         BLBC    R0,50$                         ; Br if error
50   2C A3   D0      0DA9  3436         MOVL    IRP$L_SVAPTE(R3),R0            ; Get P2 buffer address
          38   BB    0DAD  3437         PUSHR   #^M<R3,R4,R5>                  ; Save sacred registers
    62    51   28    0DAF  3438         MOVC3   R1,(R2),P2B_T_DATA(R0)         ; Save P2 char buffer
     0C A0             0DB2
          38   BA    0DB4  3439         POPR    #^M<R3,R4,R5>                  ; Restore registers
    50    01   9A    0DB6  3440 40$:    MOVZBL  S^#SS$_NORMAL,R0               ; Set success
                05   0DB9  3441 50$:    RSB                                    ; Return
```

```
                     ;+
                     ; CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
                     ;
                     ; Functional description:
                     ;
                     ; This routines checks the P1 and P2 buffers for write access if supplied.
                     ;
                     ; Inputs:
                     ;
                     ;       R1 = Size of P1 buffer needed for write access
                     ;       R3 = IRP address
                     ;       R4 = PCB address
                     ;       R5 = UCB address
                     ;       R7 = Function code
                     ;
                     ; Outputs:
                     ;
                     ;       R0 is destroyed.
                     ;       R1 = Length of P2 buffer (zero if no P2 buffer)
                     ;       R2 = Address of P2 buffer in user's process space
                     ;
                     ;
                     ;       No RETURN on NO ACCESS
                     ;
                     ; Implicit Outputs:
                     ;
                     ;       IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
                     ;
                     ;--

                     CHECK_BUFS:
                     CHECK_BUFS:
27 10                        BSBB    CHECK_P1                ; Check P1 buffer
                     CHECK_P2:
51 D4                        CLRL    R1                      ; Assume no P2 buffer desc
52 04 AC D0                  MOVL    P2(AP),R2               ; Get address of P2 desc
18 13                        BEQL    10$                     ; Br if no P2
                             IFNORD  #8,(R2),ACCESS          ; Br if no access
51 62 3C                     MOVZWL  (R2),R1                 ; Get length of buffer
0D 13                        BEQL    10$                     ; Br if zero
50 04 A2 D0                  MOVL    DSC$A_POINTER(R2),R0    ; Get buffer address
00000000'GF 16               JSB     G^EXE$READCHK           ; Check write access to buffer
                                                             ; (no return no access)
                                                             ; Also sets IRP$V_FUNC in IRP
52 50 D0                     MOVL    R0,R2                   ; Copy buffer address
                     10$:    RSB                             ; Return to caller

50 0C 9A             ACCESS: MOVZBL  S^#SS$_ACCVIO,R0        ; Return access violation
FF55D 31                     BRW     ABORTIO                 ; Abort the I/O request
```

```
                        ;  CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS

                        ;  Functional description:
                        ;
                        ;  This routine checks the P1 buffer and if okay, the buffer address
                        ;  is saved in IRP$L_MEDIA of the IRP.
                        ;
                        ;  Inputs:
                        ;
                        ;       R1 = Size of buffer for write access
                        ;       R3 = IRP address
                        ;       R4 = PCB address
                        ;       R5 = UCB address
                        ;       R7 = Function code
                        ;
                        ;  Outputs:
                        ;
                        ;       R0 is destroyed.
                        ;
                        ;       No RETURN on NO ACCESS.
                        ;
                        ;  Implicit Outputs:
                        ;
                        ;       IRP$L_MEDIA(R3) = User P1 buffer address.
                        ;       IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
                        ;
                        ;--

                        CHECK_P1:
                        CHECK_P1:
38 A3   D4              CLRL    IRP$L_MEDIA(R3)                 ; Assume no P1 buffer
50 6C   D0              MOVL    P1(AP),R0                       ; Get address of user buffer
   06   13              BEQL    10$                             ; Br if none
00000000'GF 16          JSB     G^EXE$READCHK                   ; Check access to buffer
                                                                ; (No return - no access)
3C A3   50      10$:    MOVL    R0,IRP$L_XQ_USERBUF(R3)         ; Save P1 buffer address in IRP
                05      RSB                                     ; Return to caller
```

```
                     0DF6  3532          .SBTTL  ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
                     0DF6  3533 ;++
                     0DF6  3534 ; ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
                     0DF6  3535 ;
                     0DF6  3536 ; Functional description:
                     0DF6  3537 ;
                     0DF6  3538 ; This routine allocates a system buffer and returns the address in the IRP at
                     0DF6  3539 ; IRP$L_SVAPTE.  The size of the allocation, including buffer header must
                     0DF6  3540 ; be at least 24 bytes in length.
                     0DF6  3541 ;
                     0DF6  3542 ; Inputs:
                     0DF6  3543 ;
                     0DF6  3544 ;       R1 = Size of allocation desired
                     0DF6  3545 ;       R3 = IRP address
                     0DF6  3546 ;
                     0DF6  3547 ; Outputs:
                     0DF6  3548 ;
                     0DF6  3549 ;       R0 = status of request
                     0DF6  3550 ;
                     0DF6  3551 ;       R1-R5 are preserved.
                     0DF6  3552 ;
                     0DF6  3553 ; Implicit Outputs:
                     0DF6  3554 ;
                     0DF6  3555 ;       IRP$L_SVAPTE(R3) = address of system buffer
                     0DF6  3556 ;       IRP$W_BOFF(R3) = byte count charged to user's process
                     0DF6  3557 ;       IRP$W_BCNT(R3) = original byte count requested
                     0DF6  3558 ;
                     0DF6  3559 ;       All parts of the P2 buffer header are initialized, except for the
                     0DF6  3560 ;       user's P2 buffer address.
                     0DF6  3561 ;
                     0DF6  3562 ;--
                     0DF6  3563
                     0DF6  3564 ALLOC_P2BUF:                                    ; Allocate a non-paged buffer
            51   D5  0DF6  3565          TSTL    R1                            ; Zero length buffer?
            4D   13  0DF8  3566          BEQL    30$                           ; Br if yes
            0E   BB  0DFA  3567          PUSHR   #^M<R1,R2,R3>                 ; Save registers
     32 A3  51   B0  0DFC  3568          MOVW    R1,IRP$W_BCNT(R3)             ; Save original byte count
;RNG0001     0C  51   D1  0E00    .1     CMPL    R1,#24-P2B_C_LENGTH           ; Is buffer big enough?
;RNG0001         03   1A  0E03    .2     BGTRU   5$                            ; Br if yes
;RNG0001     51  0C   D0  0E05    .3     MOVL    #24-P2B_C_LENGTH,R1           ; Else, set size to minimum
;RNG0001     51  0C   C0  0E08    .4 5$:  ADDL2   #P2B_C_LENGTH,R1              ; Add in size of header
-4   00000000'GF   16  0E0B  3573          JSB     G^EXE$BUFQUOPRC               ; Check for buffered quota
        0E  50   E9  0E11  3574          BLBC    R0,10$                        ; Branch if quota bad
                     0E14  3575 ;
                     0E14  3576 ; Quota OKAY, allocate buffer and copy info.
                     0E14  3577 ;
            51   DD  0E14  3578          PUSHL   R1                            ; Save size to charge user
   00000000'GF   16  0E16  3579          JSB     G^EXE$ALLOCBUF                ; Go allocate a buffer
        06  50   E8  0E1C  3580          BLBS    R0,20$                        ; Br if success
     5E     04   C0  0E1F  3581          ADDL    #4,SP                         ; Pop saved size
            0E   BA  0E22  3582 10$:     POPR    #^M<R1,R2,R3>                 ; Restore registers
                 05  0E24  3583          RSB                                   ; Return with error code in R0
                     0E25  3584 ;
                     0E25  3585 ; System buffer allocated decrement user's quota
                     0E25  3586 ;
            53 8ED0  0E25  3587 20$:     POPL    R3                            ; Restore user quota charge
     62  0C A2   9E  0E28  3588          MOVAB   P2B_T_DATA(R2),P2B_L_POINTER(R2) ; Set address to start of data
```

```
                08 A2    53    B0  0E2C  3589          MOVW    R3,P2B_W_SIZE(R2)           ; Save buffer size in buffer
                50       52    D0  0E30  3590          MOVL    R2,R0                       ; Save P2 char buf addr
             52 0080     C4    D0  0E33  3591          MOVL    PCB$L_JIB(R4),R2            ; Get JIB address
                20 A2    53    C2  0E38  3592          SUBL    R3,JIB$L_BYTCNT(R2)         ; Decrement user's quota
                         0E    BA  0E3C  3593          POPR    #^M<R1,R2,R3>               ; Restore registers
                2C A3    50    D0  0E3E  3594          MOVL    R0,IRP$L_SVAPTE(R3)         ; Save P2 buffer address in IRP
             30 A3    08 A0    B0  0E42  3595          MOVW    P2B_W_SIZE(R0),IRP$W_BOFF(R3) ; Return buffer size in IRP
                50       01    9A  0E47  3596 30$:     MOVZBL  S^#SS$_NORMAL,R0            ; Set success
                               05  0E4A  3597          RSB                                 ; Return to caller
```

```
                        0E4B  3599          .SBTTL  STARTIO - START I/O OPERATION
                        0E4B  3600 ;++
                        0E4B  3601 ; STARTIO - START I/O OPERATION
                        0E4B  3602 ;
                        0E4B  3603 ; Functional description:
                        0E4B  3604 ;
                        0E4B  3605 ; This routine is called when an IRP is ready to be processed by the driver.
                        0E4B  3606 ; The request is dispatched to the appropriate routine base on the internal
                        0E4B  3607 ; function code in the IRP.
                        0E4B  3608 ;
                        0E4B  3609 ; Inputs:
                        0E4B  3610 ;
                        0E4B  3611 ;       R3 = IRP address
                        0E4B  3612 ;       R5 = UCB address
                        0E4B  3613 ;
                        0E4B  3614 ;       IPL = FIPL
                        0E4B  3615 ;
                        0E4B  3616 ; Outputs:
                        0E4B  3617 ;
                        0E4B  3618 ;       R0-R2,R4 are destroyed.
                        0E4B  3619 ;--
                        0E4B  3620
                        0E4B  3621 STARTIO::                                    ; Process an I/O packet
;RNG0001        54  00C8 C5  D0  0E4B     .1          MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2               51  21 A3   9A  0E50  3624          MOVZBL  IRP$B_XQ_FUNC(R3),R1    ; Get the internal function code
                        0E54  3625 10$:     $DISPATCH       R1,TYPE=B,-
                        0E54  3626          <-      ;function       action
                        0E54  3627
                        0E54  3628                  <XQ_FC_V_INIT   STARTUP>,-      ; Startup request
                        0E54  3629                  <XQ_FC_V_STOP   SHUT>,-         ; Shutdown request
                        0E54  3630                  <XQ_FC_V_CHMODE CHMODE>,-       ; Set new multicast list
                        0E54  3631          >
                        0E66  3632 ;
                        0E66  3633 ; Other request type
                        0E66  3634 ;
                        0E66  3635          BUG_CHECK NOBUFPCKT,FATAL       ; Fatal error
                        0E6A  3636
                        0E6A  3637 ;
                        0E6A  3638 ; Startup unit's protocol
                        0E6A  3639 ;
                        0E6A  3640 STARTUP:                                     ; Startup unit's protocol
            49  10      0E6A  3641          BSBB    START                       ; Start protocol
         01 50  E9      0E6C  3642          BLBC    R0,10$                      ; Br if error on startup
                05      0E6F  3643          RSB                                 ; Else, return to caller
                        0E70  3644
            50  DD      0E70  3645 10$:     PUSHL   R0                          ; Save error return
            2A  10      0E72  3646          BSBB    STOP                        ; Shutdown unit
            50  8ED0    0E74  3647          POPL    R0                          ; Restore error return
          0CD6  31      0E77  3648          BRW     IO_DONE                     ; Complete the I/O request
                        0E7A  3649
                        0E7A  3650 ;
                        0E7A  3651 ; Shutdown UNIT's protocol
                        0E7A  3652 ;
                        0E7A  3653 SHUT:                                        ; Shutdown protocol
0210 C4     55  D1      0E7A  3654          CMPL    R5,CDB_L_PRMUSER(R4)        ; Are we the PROMISCUOUS user?
            14  12      0E7F  3655          BNEQ    10$                         ; Br if not
      0210 C4   D4      0E81  3656          CLRL    CDB_L_PRMUSER(R4)           ; Else, clear the PROMISCUOUS user
```

```
                01   90  0E85  3657          MOVB    #NMA$C_STATE_OFF,-      ; Don't forget about the CDB
          0247 C4        0E87  3658                  CDB_B_PRM(R4)           ;  parameter
             04D7   30   0E8A  3659          BSBW    SETUP_MODE              ; Get setup buffer
           0D 50    E9   0E8D  3660          BLBC    R0,90$                  ; Br if error
              03    90   0E90  3661          MOVB    #XQ_FC_V_STOP,-         ; Set function request
           20 A2        0E92  3662                  CXB$B_XQ_FUNC(R2)       ;
                   05   0E94  3663          RSB                             ; Return to complete function
                        0E95  3664
                07   10   0E95  3665 10$:    BSBB    STOP                    ; Shutdown unit
          50    01   9A   0E97  3666          MOVZBL  S^#SS$_NORMAL,R0        ; Return success
             0CB3   31   0E9A  3667          BRW     IO_DONE                 ; Complete I/O request
                        0E9D  3668
                   05   0E9D  3669 90$:     RSB                             ; Return to caller
                        0E9E  3670 ;
                        0E9E  3671 ; STOP the unit
                        0E9E  3672 ;
                        0E9E  3673 STOP:                                    ; Stop the protocol
:RNG0001        54  00C8 C5   D0   0E9E    .1           MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2                    1178   31   0EA3  3676          BRW     SHUTDOWN_PROTYP         ; Shutdown the unit
                        0EA6  3677 ;
                        0EA6  3678 ; CHMODE - set new multicast list
                        0EA6  3679 ;
                        0EA6  3680 CHMODE:
                   04BB 30   0EA6  3681          BSBW    SETUP_MODE              ; Get XMIT setup buffer
                 06 50  E9   0EA9  3682          BLBC    R0,10$                  ; Exit if error
                    06  90   0EAC  3683          MOVB    #XQ_FC_V_CHMODE,-       ; Set function request
                 20 A2        0EAE  3684                  CXB$B_XQ_FUNC(R2)       ;
                    9E  17   0EB0  3685          JMP     @(SP)+                  ; Call back caller and return
                        0EB2  3686
                   0C9B 31   0EB2  3687 10$:     BRW     IO_DONE                 ; Complete I/O request
```

```
                     0EB5  3689          .SBTTL  START - START UNIT'S PROTOCOL
                     0EB5  3690 ;++
                     0EB5  3691 ; START - START UNIT'S PROTOCOL
                     0EB5  3692 ;
                     0EB5  3693 ; Functional description:
                     0EB5  3694 ;
                     0EB5  3695 ; This routine initiates the protocol on the unit. The QNA is reset if first
                     0EB5  3696 ; unit online.  The free list is filled and the first receive started.
                     0EB5  3697 ; If a failure occurs the unit shutdown sequence is entered.
                     0EB5  3698 ;
                     0EB5  3699 ; Inputs:
                     0EB5  3700 ;
                     0EB5  3701 ;       R3 = IRP address
                     0EB5  3702 ;       R5 = UCB address
                     0EB5  3703 ;
                     0EB5  3704 ;       IPL = FIPL
                     0EB5  3705 ;
                     0EB5  3706 ; Implicit inputs:
                     0EB5  3707 ;
                     0EB5  3708 ;       IRP$L_MEDIA contains a copy of the mode buffer specified by the user.
                     0EB5  3709 ;       IRP$W_BOFF contains the quota taken from the user for the unit.
                     0EB5  3710 ;
                     0EB5  3711 ; Outputs:
                     0EB5  3712 ;
                     0EB5  3713 ;       R0 = Status return for startup request.
                     0EB5  3714 ;       R1,R2,R4 are destroyed.
                     0EB5  3715 ;       R3,R5 are preserved.
                     0EB5  3716 ;
                     0EB5  3717 ;--
                     0EB5  3718
                     0EB5  3719 START::                                      ; Start protocol operation
          00   91    0EB5  3720          CMPB    #NMA$C_LINPR_POI,-          ;% Are we in PT-TO-PT mode?
     00DC C5         0EB7  3721                  UCB$B_XQ_PRO(R5)            ;%
          24   12    0EBA  3722          BNEQ    3$                          ;% Br if not
51   00C4 8F   3C    0EBC  3723          MOVZWL  #IRP$C_LENGTH,R1            ;% Set size of an IRP
          53   DD    0EC1  3724          PUSHL   R3                          ;% Save R3
 00000000'GF   16    0EC3  3725          JSB     G^EXE$ALONONPAGED           ;% Allocate the IRP
          53 8ED0    0EC9  3726          POPL    R3                          ;% Restore R3
        06   50   E8 0ECC  3727          BLBS    R0,1$                       ;% Br if success
50   0124 8F   3C    0ECF  3728          MOVZWL  #SS$_INSFMEM,R0             ;% Set error return
          05   0ED4  3729          RSB                                     ;% Return to caller
                     0ED5  3730
0196 C5   52   D0    0ED5  3731 1$:      MOVL    R2,UCB$L_XQ_STIRP(R5)       ;% Save startup IRP
     0060 8F   A8    0EDA  3732          BISW    #UCB$M_XQ_START!UCB$M_XQ_STACK,- ;% We are now in the startup
       68 A5         0EDE  3733                  UCB$W_DEVSTS(R5)            ;%   and stack wait state
                     0EE0  3734
     00BC C5   D5    0EE0  3735 3$:      TSTL    UCB$L_XQ_CPID(R5)           ; Creator PID saved already?
          06   12    0EE4  3736          BNEQ    5$                          ; Br if yes
       20 A5   D0    0EE6  3737          MOVL    UCB$L_CPID(R5),UCB$L_XQ_CPID(R5) ; Else, save creator PID
     00BC C5         0EE9
                     0EEC  3738 ;
                     0EEC  3739 ; Set up idle UCB
                     0EEC  3740 ;
     0800 8F   3C    0EEC  3741 5$:      MOVZWL  #XM$M_STS_ACTIVE,-
       44 A5         0EF0  3742                  UCB$L_DEVDEPEND(R5)         ; Reset status and error summary
  21 A3   05   91    0EF2  3743          CMPB    #XQ_FC_V_RESTART,IRP$B_XQ_FUNC(R3) ; Is this a re-start operation?
          06   13    0EF6  3744          BEQL    8$                          ; Br if yes - don't reset the PID
```

```
                      0C A3   D0  0EF8  3745          MOVL    IRP$L_PID(R3),UCB$L_XQ_PID(R5) ; Save starter's PID
                  00B8 C5         0EFB
                                  0EFE  3746 ;
                                  0EFE  3747 ; Check for CDB
                                  0EFE  3748 ;
:RNG0001          52    00C8 C5   D0  0EFE    .1 8$:    MOVL    UCB$L_XQ_CDB(R5),R2      ; Get CDB address
:RNG0001                0208 C2   96  0F03    .2        INCB    CDB_B_UNTCNT(R2)         ; One more unit on this controller
:RNG0001                          0F07    .3        ASSUME  CDB_STS_V_INITED  EQ  0
:RNG0001            03  0246 C2   E8  0F07    .4        BLBS    CDB_B_STS(R2),10$        ; Br if already inited
-5                        0093    31  0F0C  3754        BRW     20$                      ; Else, init CDB
                                  0F0F  3755
                      54    52    D0  0F0F  3756 10$:     MOVL    R2,R4                    ; Copy CDB address
                  50  00E1 C5    9E  0F12  3757          MOVAB   UCB$B_XQ_CDBPRM(R5),R0   ; Get UCB parameter address
                  51  0249 C4    9E  0F17  3758          MOVAB   CDB_B_SETPRM(R4),R1      ; Get CDB parameter address
                      52    01    9A  0F1C  3759          MOVZBL  #UCB$C_XQ_CDBPRM,R2      ; Set size of parameter list
                                  0F1F  3760
                                  0F1F  3761 ;
                                  0F1F  3762 ; Check order of UCB parameters
                                  0F1F  3763 ;
                                  0F1F  3764          ASSUME  UCB$B_XQ_CON EQ UCB$B_XQ_CDBPRM
                                  0F1F  3765 ;
                                  0F1F  3766 ; Check order of CDB parameters
                                  0F1F  3767 ;
                                  0F1F  3768          ASSUME  CDB_B_CON EQ CDB_B_SETPRM
                                  0F1F  3769
                                  0F1F  3770          ASSUME  NMA$C_STATE_ON NE -1
                                  0F1F  3771          ASSUME  NMA$C_STATE_OFF NE -1
                      61    60    91  0F1F  3772 13$:     CMPB    (R0),(R1)                ; Match?
                            73    12  0F22  3773          BNEQ    12$                      ; Br if no
                      80    81    90  0F24  3774          MOVB    (R1)+,(R0)+              ; Store CDB value in UCB
                        F5  52    F5  0F27  3775          SOBGTR  R2,13$                   ; Loop if more to check
                                  0F2A  3776 ;
                                  0F2A  3777 ; NOW, check if user has given a hardware physical address.
                                  0F2A  3778 ;
                                  0F2A  3779 ;       R0 = Address of parameters in UCB
                                  0F2A  3780 ;       R1 = Address of parameters in CDB
                                  0F2A  3781 ;
                                  0F2A  3782          ASSUME  CDB_G_PHA EQ CDB_B_CON+1
                                  0F2A  3783          ASSUME  UCB$G_XQ_PHA EQ UCB$B_XQ_CON+1
                  FFFFFFFF  8F    D1  0F2A  3784          CMPL    #-1,(R0)                 ; Is user physical address defined?
                            60        0F30
                            08    12  0F31  3785          BNEQ    15$                      ; Br if yes
                        FFFF  8F    B1  0F33  3786          CMPW    #-1,4(R0)                ; Is user physical address defined?
                          04  A0        0F37
                            10    13  0F39  3787          BEQL    16$                      ; Br if not
                      0B04  8F    B0  0F3B  3788 15$:     MOVW    #NMA$C_PCLI_PHA,IRP$W_XQ_CODE(R3) ; Assume bad physical address
                          40  A3        0F3F
                      81    80    D1  0F41  3789          CMPL    (R0)+,(R1)+              ; Physical address match??
                            58    12  0F44  3790          BNEQ    19$                      ; Br if no
                      81    80    B1  0F46  3791          CMPW    (R0)+,(R1)+              ; Still match??
                            53    12  0F49  3792          BNEQ    19$                      ; Br if no
                      024A  C4    D0  0F4B  3793 16$:     MOVL    CDB_G_PHA(R4),-          ; Return hardware set address
                      00E2  C5        0F4F  3794                   UCB$G_XQ_PHA(R5)        ;  just in case we defaulted
                      024E  C4    B0  0F52  3795          MOVW    CDB_G_PHA+4(R4),-        ;       ...
                      00E6  C5        0F56  3796                   UCB$G_XQ_PHA+4(R5)      ;       ...
                                  0F59  3797 ;
                                  0F59  3798 ; Check users buffer size - must not be more than twice the hardware buffer
```

```
                                0F59  3799 ; size. (Already has been checked against max message size).
                                0F59  3800 ;
      0AF1 8F    B0             0F59  3801          MOVW    #NMA$C_PCLI_BUS,IRP$W_XQ_CODE(R3) ; Assume bad buffer size
        40 A3                   0F5D
   51   0108 C4   3C             0F5F  3802          MOVZWL  CDB_W_BSZ(R4),R1          ; Get device buffer size
     05 00E0 C5   E8             0F64  3803          BLBS    UCB$B_XQ_DCH(R5),17$      ; Br if user can't do data chaining
                                0F69  3804                                            ; && Maybe this is an Internal IRP user
   51   0108 C4   A0             0F69  3805          ADDW    CDB_W_BSZ(R4),R1          ; Compute twice the normal buffer size
     51   42 A5   B1             0F6E  3806 17$:     CMPW    UCB$W_DEVBUFSIZ(R5),R1    ; Is buffer size okay?
             2A   1A             0F72  3807          BGTRU   19$                       ; Br if too large
          40 A3   B4             0F74  3808          CLRW    IRP$W_XQ_CODE(R3)         ; No bad parameters
             10   A8             0F77  3809          BISW    #UCB$M_XQ_RUN,-           ; Indicate we have entered RUN mode
          68 A5                 0F79  3810                  UCB$W_DEVSTS(R5)          ;
           1589   30             0F7B  3811          BSBW    MOVE_MULTI                ; Copy multicast address list
                                0F7E  3812          ASSUME  NMA$C_STATE_ON EQ 0
                                0F7E  3813          ASSUME  NMA$C_STATE_OFF EQ 1
     07 00DE C5   E8             0F7E  3814          BLBS    UCB$B_XQ_PRM(R5),173$     ; Br if not promiscuous
        00DE C5   90             0F83  3815          MOVB    UCB$B_XQ_PRM(R5),-        ; Else, enable promiscuous mode
        0247 C4                 0F87  3816                  CDB_B_PRM(R4)
           03D7   30             0F8A  3817 173$:    BSBW    SETUP_MODE                ; Allocate setup mode buffer
          06 50   E9             0F8D  3818          BLBC    R0,175$                   ; Exit if error
             06   90             0F90  3819          MOVB    #XQ_FC_V_CHMODE,-         ; Set function request
          20 A2                 0F92  3820                  CXB$B_XQ_FUNC(R2)         ;
              9E   16             0F94  3821          JSB     @(SP)+                    ; Complete request
                   05             0F96  3822 175$:    RSB                               ; Return to caller
                                0F97  3823 ;
                                0F97  3824 ; Error on parameter validation
                                0F97  3825 ;
      F208 CF42    B0             0F97  3826 18$:     MOVW    BAD_PARAM_TBL-2[R2],IRP$W_XQ_CODE(R3) ; Return parameter code
        40 A3                   0F9C
        50   14   9A             0F9E  3827 19$:     MOVZBL  S^#SS$_BADPARAM,R0        ; Return bad parameter error
                   05             0FA1  3828          RSB                               ; Return to caller
                                0FA2  3829 ;
                                0FA2  3830 ; Initialize CDB
                                0FA2  3831 ;
   51   020A 8F    3C             0FA2  3832 20$:     MOVZWL  #CDB_C_ZERO,R1            ; Get portion of CDB to init with zero
             3E   BB             0FA7  3833          PUSHR   #^M<R1,R2,R3,R4,R5>       ; Save registers
   00   62   00   2C             0FA9  3834          MOVC5   #0,(R2),#0,R1,(R2)        ; Zero the structure
        62   51                 0FAD
             3E   BA             0FAF  3835          POPR    #^M<R1,R2,R3,R4,R5>       ; Restore registers
                                0FB1  3836          ASSUME  CDB_L_FQFL EQ 0
                                0FB1  3837          ASSUME  CDB_L_FQBL EQ CDB_L_FQFL+4
        54   82   7E             0FB1  3838          MOVAQ   (R2)+,R4                  ; Skip link pointers, copy CDB address
                                0FB4  3839          ASSUME  CDB_W_SIZE EQ CDB_L_FQBL+4
        82   51   B0             0FB4  3840          MOVW    R1,(R2)+                  ; Store size of structure
                                0FB7  3841          ASSUME  CDB_B_TYPE EQ CDB_W_SIZE+2
                                0FB7  3842          ASSUME  CDB_B_FIPL EQ CDB_B_TYPE+1
   82   0833 8F    B0             0FB7  3843          MOVW    #<IPL$_XQ_FIPL@8>!DYN$C_CDB,(R2)+ ; Set structure type and FIPL
                                0FBC  3844          ASSUME  CDB_L_FPC EQ CDB_B_FIPL+1
    00001663'EF   9E             0FBC  3845          MOVAB   FORK_PROC,(R2)+           ; Set fork process address
                   82             0FC2
                                0FC3  3846          ASSUME  CDB_L_FR3 EQ CDB_L_FPC+4
                                0FC3  3847          ASSUME  CDB_L_FR4 EQ CDB_L_FR3+4
                   82   7C       0FC3  3848          CLRQ    (R2)+                     ; Clear fork R3 and R4
                                0FC5  3849          ASSUME  CDB_B_NEXTXMT EQ CDB_L_FR4+4
                                0FC5  3850          ASSUME  CDB_B_NEXTRCV EQ CDB_B_NEXTXMT+1
                                0FC5  3851          ASSUME  CDB_B_RCVMAP EQ CDB_B_NEXTRCV+1
```

```
                                0FC5  3852          ASSUME  CDB_B_XMTMAP EQ CDB_B_RCVMAP+1
                          82    D4  0FC5  3853          CLRL    (R2)+                   ; Clear slot in use flags
                                0FC7  3854          ASSUME  CDB_L_RCVMAP EQ CDB_B_XMTMAP+1
                                0FC7  3855          ASSUME  CDB_L_XMTMAP EQ CDB_L_RCVMAP+<4*<MAX_C_RCV-1>>
                          51    0A  9A  0FC7  3856          MOVZBL  #<MAX_C_XMT-1>+<MAX_C_RCV-1>,R1 ; Set number of mapping vectors
                          82    01  CE  0FCA  3857 30$:     MNEGL   #1,(R2)+                ; Indicate no mapping info
                            FA  51  F5  0FCD  3858          SOBGTR  R1,30$                  ; Loop if more
                                0FD0  3859          ASSUME  CDB_L_RRINGPA EQ CDB_L_XMTMAP+<4*<MAX_C_XMT-1>>
                    00000098    8F  C0  0FD0  3860          ADDL    #CDB_Q_QUEUES*CDB_L_RRINGPA,R2  ; Skip ring entry pointers
                                52  0FD6
                          51    05  9A  0FD9  3861          MOVZBL  #CDB_C_QUEUES,R1        ; Set number of queue listheads
                          82    62  DE  0FDC  3862 40$:     MOVAL   (R2),(R2)+              ; Set forward link
                        82      FC  A2  0FDF  3863          MOVL    -4(R2),(R2)+            ; Set backward link
                            F6  51  F5  0FE1  3864          SOBGTR  R1,40$                  ; Loop if more listheads
;RNG0001            0246    C4  01  88  0FE4    .1          BISB    #CDB_STS_M_INITED,CDB_B_STS(R4) ; Set initial status bits
-1
                                0FE9  3866 ;
                                0FE9  3867 ; Initialize CDB defaults
                                0FE9  3868 ;
                          00DA  C5  B0  0FE9  3869          MOVW    UCB$W_XQ_BSZ(R5),-      ; Init CDB buffer size
                          0108  C4        0FED  3870                  CDB_W_BSZ(R4)           ;
                          00D6  C5  B0  0FF0  3871          MOVW    UCB$W_XQ_HBQ(R5),CDB_W_QUOTA(R4) ; Set initial quota
                          010A  C4        0FF4
                                0FF7  3872
                                0FF7  3873          ASSUME  INIT_C_AQUOTA LE 255
                                0FF7  3874          ASSUME  CDB_B_AQUOTA EQ CDB_B_AQUOTA+1
                          0200  8F  B0  0FF7  3875          MOVW    #<INIT_C_AQUOTA@8>,-    ; Initialize Maximum QUOTA
                          02EE  C4        0FFB  3876                   CDB_B_AQUOTA(R4)       ; and zero Additional QUOTA
                                0FFE  3877          ASSUME  UCB$B_XQ_MLT EQ UCB$B_XQ_PRM+1
                                0FFE  3878          ASSUME  CDB_B_MLT EQ CDB_B_PRM+1
                          00DE  C5  B0  0FFE  3879          MOVW    UCB$B_XQ_PRM(R5),CDB_B_PRM(R4) ; Set the promiscuous mode
                          0247  C4        1002
                                1005  3880                                              ;  and the all multicast enable
                    50    00E1  C5  9E  1005  3881          MOVAB   UCB$B_XQ_CDBPRM(R5),R0  ; Get address of UCB parameters
                    51    0249  C4  9E  100A  3882          MOVAB   CDB_B_SETPRM(R4),R1     ; Get address of CDB parameters
                          52    07  9A  100F  3883          MOVZBL  #UCB$C_XQ_SETPRM,R2     ; Set size of parameter list
                          81    80  90  1012  3884 45$:     MOVB    (R0)+,(R1)+             ; Store parameters
                            FA  52  F5  1015  3885          SOBGTR  R2,45$                  ; Loop if more
                              14EC  30  1018  3886          BSBW    MOVE_MULTI              ; Copy multicast address list
                          0108  C4  B0  101B  3887          MOVW    CDB_Q_BSZ(R4),-         ; Set buffer size
                            7E  A5        101F  3888                  UCB$W_BCNT(R5)
                                1021  3889          PUSHQ   R6                      ; Save R6,R7
                          56    54  D0  1024  3890          MOVL    R4,R6                   ; Save CDB address
                        54      24  A5  D0  1027  3891          MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
;RNG0001                         102B    .1          ASSUME  IDB$L_CSR  EQ  0
;RNG0001                   2C    B4  D0  102B    .2          MOVL    @CRB$L_INTD+VEC$L_IDB(R4),- ; Save CSR address
;RNG0001             0114  C6        102F    .3                  CDB_L_CSR(R6)
                        57      28  A5  D0  1031  3892          MOVL    UCB$L_DDB(R5),R7        ; Get DDB address
                          04    A7  D0  1035  3893          MOVL    DDB$L_UCB(R7),CDB_L_UCB0(R6) ; Set UCB #0 address
                          0110  C6        1038
                                103B  3894
                                103B  3895          CPUDISP <<790,50$>,-
                                103B  3896                   <780,50$>,-
                                103B  3897                   <750,50$>,-
                                103B  3898                   <730,50$>,-
;RNG0001                         103B    .1                   <UV2,50$>,-
;RNG0001                         103B    .2                   <UV1,70$>>
;RNG0001                         1057    .3
```

```
;RNG0001                                1057    .4 50$:     ;  All but MicroVAX I
;RNG0001                                1057    .5
;RNG0001                                1057    .6          ;
-2                                      1057  3901          ; Allocate map registers for receive buffers and one transmit buffer.
                                        1057  3902          ;
                    01FF 8F     B0      1057  3903          MOVW    #511,UCB$W_BOFF(R5)     ; Set worst case byte offset
                      7C A5             105B
                                        105D  3904          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                        105D  3905          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                       34 A4    D4      105D  3906          CLRL    CRB$L_INTD+VEC$W_MAPREG(R4) ; Clear map register + datapath
                   56   1C A6    DE      1060  3907          MOVAL   CDB_L_RCVMAP(R6),R6     ; Get mapping slot address
                        57   07    9A      1064  3908          MOVZBL  #MAX_C_RCV-1,R7         ; Get number of receive slots
                00000000'GF     16      1067  3909 55$:     JSB     G^IOC$ALOUBAMAP         ; Allocate a set of map registers
                       39 50    E9      106D  3910          BLBC    R0,60$                  ; Br if unavailable
                   86   34 A4    D0      1070  3911          MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),(R6)+ ; Save map info
                       F0 57    F5      1074  3912          SOBGTR  R7,55$                  ; Continue
                                        1077  3913          ;
                    05EE 8F     3C      1077  3914          MOVZWL  #MAX_PKT_SIZE+18,-      ; Set transmit buffer size to max
                      7E A5             107B  3915                  UCB$W_BCNT(R5)          ;  Ethernet packet size + header
                00000000'GF     16      107D  3916          JSB     G^IOC$ALOUBAMAP         ; Allocate a set of transmit registers
                       23 50    E9      1083  3917          BLBC    R0,60$                  ; Br if unavailable
                                        1086  3918          ASSUME  CDB_L_XMTMAP EQ CDB_L_RCVMAP+<4*<MAX_C_RCV-1>>
                   66   34 A4    D0      1086  3919          MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),(R6) ; Save transmit map info
                                        108A  3920          ;
                                        108A  3921          ; Allocate mapping for QNA RING structures.
                                        108A  3922          ;
                   56   10 A4    D0      108A  3923          MOVL    CRB$L_AUXSTRUC(R4),R6   ; Get CDB address
                                        108E  3924
                                        108E  3925          ASSUME  CDB_G_MAPPED EQ CDB_G_RRING
                                        108E  3926          ASSUME  CDB_G_XRING EQ CDB_G_RRING+RCV_K_LENGTH
                57   0162 C6    9E      108E  3927          MOVAB   CDB_G_MAPPED(R6),R7     ; Get starting ring address
                       FE00 8F    AB      1093  3928          BICW3   #^C<VA$M_BYTE>,-        ; Get PCBB byte offset
                     7C A5   57          1097  3929                  R7,UCB$W_BOFF(R5)
                       00A8 8F    B0      109A  3930          MOVW    #CDB_C_MAPPED,UCB$W_BCNT(R5) ; Set Block size
                         7E A5           109E
                 00000000'GF     16      10A0  3931          JSB     G^IOC$ALOUBAMAP         ; Allocate map registers
                        09 50    E8      10A6  3932          BLBS    R0,65$                  ; Br if allocated
                                        10A9  3933 60$:     POPQ    R6                      ; Restore R6, R7
                50   0344 8F    3C      10AC  3934          MOVZWL  #SS$_INSFMAPREG,R0      ; Set insufficient map registers
                                    05  10B1  3935          RSB                             ; Return with error
                                        10B2  3936
                                        10B2  3937          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                                        10B2  3938          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                       34 A4    D0      10B2  3939 65$:     MOVL    CRB$L_INTD+VEC$W_MAPREG(R4),- ; Save RING mapping info
                     011C C6           10B5  3940                  CDB_L_RINGMAP(R6)       ;
                                        10B8  3941          ;
                                        10B8  3942          ; Convert the virtual RING address to a UNIBUS mapped address
                                        10B8  3943          ;
                          09    EF      10B8  3944          EXTZV   S^#VA$V_VPN,-           ; Get RING page number
                51   57   15            10BA  3945                  S^#VA$S_VPN,R7,R1       ;
                 00000000'GF     D0      10BD  3946          MOVL    G^MMG$GL_SPTBASE,R0     ; Get SPT address
                                50      10C3
                78 A5    6041    DE      10C4  3947          MOVAL   (R0)[R1],UCB$L_SVAPTE(R5) ; Set PTE address
                 00000000'GF     16      10C9  3948          JSB     G^IOC$LOADUBAMAP        ; Load the PCBB map registers
;RNG0001           51   7C A5    3C      10CF    .1          MOVZWL  UCB$W_BOFF(R5),R1       ; Set BA0-BA8
;RNG0001                 34 A4    F0      10D3    .2          INSV    CRB$L_INTD+VEC$W_MAPREG(R4),- ; Set BA9-BA15
;RNG0001           51   0D   09          10D6    .3                  #9,#13,R1               ;
```

```
;RNG0001                22   11  10D9     .4          BRB     80$                     ; Continue
;RNG0001                             10DB     .5
;RNG0001                             10DB     .6 70$:     ;  MicroVAX I
;RNG0001                             10DB     .7
;RNG0001                             10DB     .8          ;
;RNG0001                             10DB     .9          ; Compute physical address to start of rings for MicroVAX I.
-10                                  10DB  3959           ;
                                     10DB  3960           ASSUME  CDB_G_MAPPED EQ CDB_G_RRING
                                     10DB  3961           ASSUME  CDB_G_XRING EQ CDB_G_RRING+RCV_K_LENGTH
                57   0162 C6   9E   10DB  3962           MOVAB   CDB_G_MAPPED(R6),R7     ; Get starting ring address
                57   15   09   EF   10E0  3963           EXTZV   #VA$V_VPN,#VA$S_VPN,R7,R1 ; Get virtual page number
                               51   10E4
                 00000000'GF   D0   10E5  3964           MOVL    G^MMG$GL_SPTBASE,R0     ; Get base address of SPTs
                               50   10EB
                   50    6041   D0   10EC  3965           MOVL    (R0)[R1],R0             ; Get the PTE contents
                 FFFFFE00 8F   CB   10F0  3966           BICL3   #^C<VA$M_BYTE>,R7,R1    ; Get buffer offset (BA00-BA08)
                      51   57        10F6
                                     10F8  3967           ASSUME  PTE$S_PFN GE 13
                0D   09   50   F0   10F8  3968           INSV    R0,#9,#13,R1            ; Copy BA09-BA21
                          51         10FC
                                     10FD  3969
                                     10FD  3970 80$:      ;
                                     10FD  3971           ; Now calculate the physical/virtual address of each of the ring buffer
                                     10FD  3972           ; entries, saving the addresses in the PHYSICAL ADDRESS/VIRTUAL
                                     10FD  3973           ; ADDRESS VECTOR
                                     10FD  3974           ;
                50   0162 C6   9E   10FD  3975           MOVAB   CDB_G_RRING(R6),R0      ; Get first ring entry
                          57   D4   1102  3976           CLRL    R7                      ; Use R7 as ring index
             44 A647      51   D0   1104  3977 90$:      MOVL    R1,CDB_L_RRINGPA(R6)[R7]; Save RING physical address
                   51     0C   C0   1109  3978           ADDL    #RCV_C_LENGTH,R1        ; Skip to next
             7C A647      50   D0   110C  3979           MOVL    R0,CDB_L_RRINGVA(R6)[R7]; Save RING entry virtual address
                   50     0C   C0   1111  3980           ADDL    #RCV_C_LENGTH,R0        ; Skip to next
                EC 57     08   F2   1114  3981           AOBLSS  #MAX_C_RCV,R7,90$       ; Loop if more
                                     1118  3982
                                     1118  3983           ASSUME  CDB_G_XRING EQ CDB_G_RRING+<<MAX_C_RCV+1>*RCV_C_LENGTH>
             44 A647      51   D0   1118  3984           MOVL    R1,CDB_L_RRINGPA(R6)[R7]; Save RING physical address
                   51     0C   C0   111D  3985           ADDL    #RCV_C_LENGTH,R1        ; Skip over chained entry
                   50     0C   C0   1120  3986           ADDL    #RCV_C_LENGTH,R0        ; Skip over chained entry
                                     1123  3987
                          57   D4   1123  3988           CLRL    R7
             68 A647      51   D0   1125  3989 100$:     MOVL    R1,CDB_L_XRINGPA(R6)[R7]; Save RING physical address
                   51     0C   C0   112A  3990           ADDL    #XMT_C_LENGTH,R1        ; Skip to next
          009C C647       50   D0   112D  3991           MOVL    R0,CDB_L_XRINGVA(R6)[R7]; Save RING entry virtual address
                   50     0C   C0   1133  3992           ADDL    #XMT_C_LENGTH,R0        ; Skip to next
                EB 57     04   F2   1136  3993           AOBLSS  #MAX_C_XMT,R7,100$      ; Loop if more
             68 A647      51   D0   113A  3994           MOVL    R1,CDB_L_XRINGPA(R6)[R7]; Save last RING physical address
                                     113F  3995           ;
                                     113F  3996           ; Initialize receive ring buffer entries
                                     113F  3997           ;
                   57     08   9A   113F  3998           MOVZBL  #MAX_C_RCV,R7           ; Set number of entries in ring
                51   0162 C6   9E   1142  3999           MOVAB   CDB_G_RRING(R6),R1      ; Get address of RING buffer
                     8000 8F   B0   1147  4000 110$:      MOVW    #RCV_FLG_M_LAST,-       ; Init flags
                          61         114B  4001                   RCV_W_FLAG(R1)          ;
                        08 A1   B4   114C  4002           CLRW    RCV_W_STS(R1)           ; Zero status
                   51     0C   C0   114F  4003           ADDL    #RCV_C_LENGTH,R1        ; Skip to next entry
                     F2 57     F5   1152  4004           SOBGTR  R7,110$                 ; Loop if more
                                     1155  4005           ;
```

```
                                1155  4006          ; The last entry "chains" back to the first
                                1155  4007          ;
                8000 8F   B0    1155  4008          MOVW    #RCV_FLG_M_LAST,-       ; Init flags
                     61         1159  4009                  RCV_B_FLAG(R1)          ;
                  44 A6   B0    115A  4010          MOVW    CDB_L_RRINGPA(R6),-     ; Set the chain address
                  04 A1         115D  4011                  RCV_W_ADDR(R1)          ;
                  46 A6   9B    115F  4012          MOVZBW  CDB_L_RRINGPA+2(R6),-   ; Set high part of chain address
                  02 A1         1162  4013                  RCV_W_ADDRHI(R1)        ;
                          A8    1164  4014          BISW    #RCV_DSC_M_CHAIN!-      ; Indicate chain operation
                                1165  4015                   RCV_DSC_M_VALID,-     ;   and valid address
                C000 8F         1165  4016                   RCV_W_ADDRHI(R1)      ;
                  02 A1         1168
                                116A  4017          ;
                                116A  4018          ; Initialize transmit ring buffer entries
                                116A  4019          ;
                57    04  9A    116A  4020          MOVZBL  #MAX_C_XMT,R7           ; Set number of XMIT entries
          51    01CE  C6  9E    116D  4021          MOVAB   CDB_G_XRING(R6),R1      ; Get address of RING buffer
                8000 8F   B0    1172  4022 120$:    MOVW    #XMT_FLG_M_LAST,-       ; Init flags
                     61         1176  4023                  XMT_B_FLAG(R1)          ;
                  08 A1   B4    1177  4024          CLRW    XMT_W_STS(R1)           ; Zero status
                51    0C  C0    117A  4025          ADDL    #XMT_C_LENGTH,R1        ; Skip to next entry
                  F2 57   F5    117D  4026          SOBGTR  R7,120$                 ; Loop if more
                                1180  4027          ;
                                1180  4028          ; The last entry "chains" back to the first
                                1180  4029          ;
                8000 8F   B0    1180  4030          MOVW    #XMT_FLG_M_LAST,-       ; Init flags
                     61         1184  4031                  XMT_B_FLAG(R1)          ;
                  68 A6   B0    1185  4032          MOVW    CDB_L_XRINGPA(R6),-     ; Set the chain address
                  04 A1         1188  4033                  XMT_W_ADDR(R1)          ;
                  6A A6   9B    118A  4034          MOVZBW  CDB_L_XRINGPA+2(R6),-   ; Set high part of chain address
                  02 A1         118D  4035                  XMT_W_ADDRHI(R1)        ;
                          A8    118F  4036          BISW    #XMT_DSC_M_CHAIN!-      ; Indicate chain operation
                                1190  4037                   XMT_DSC_M_VALID,-     ;   and valid address
                C000 8F         1190  4038                   XMT_W_ADDRHI(R1)      ;
                  02 A1         1193
                                1195  4039 ;
;RNG0001                         1195    .1 ; Initialize contiguous buffer area for MicroVAX I.
;RNG0001                         1195    .2 ;
;RNG0001                         1195    .3          CPUDISP <<790,140$>,-
;RNG0001                         1195    .4                  <780,140$>,-
;RNG0001                         1195    .5                  <750,140$>,-
;RNG0001                         1195    .6                  <730,140$>,-
;RNG0001                         1195    .7                  <UV2,140$>,-
;RNG0001                         1195    .8                  <UV1,122$>>             ; Initialize buffer area for UV1
;RNG00C1                         11B1    .9
;RNG0001                         11B1   .10 122$:    ;  MicroVAX I
;RNG0001                         11B1   .11
;RNG0001                         11B1   .12          ;
;RNG0001                         11B1   .13          ; Init MicroVAX I buffer area
;RNG0001                         11B1   .14          ;
-9              52    020C C6   D0    11B1  4049          MOVL    CDB_L_UV1BUF(R6),R2     ; Get buffer area address
                          09    12    11B6  4050          BNEQ    123$                    ; Br if present
                                      11B8  4051          POPQ    R6                      ; Restore R6, R7
                50    0124 8F   3C    11BB  4052          MOVZWL  #SS$_INSFMEM,R0         ; Else, return error
                                05    11C0  4053          RSB
                                      11C1  4054
                                      11C1  4055 123$:    ;
```

```
                                11C1  4056          ; Compute physical/virtual address of buffers in buffer area
;RNG0001                        11C1     1          ;
                  52   0C A2  9E  11C1  4057          MOVAB   12(R2),R2                   ; Get start of buffer area
                52   15   09  EF  11C5  4058          EXTZV   #VA$V_VPN,#VA$S_VPN,R2,R0   ; Get virtual page number
                             50  11C9
                 00000000'GF  D0  11CA  4059          MOVL    G^MMG$GL_SPTBASE,R1         ; Get base address of SPTs
                             51  11D0
                   50   6140  D0  11D1  4060          MOVL    (R1)[R0],R0                 ; Get the PTE contents
                 FFFFFE00 8F  CB  11D5  4061          BICL3   #^C<VA$M_BYTE>,R2,R1        ; Get buffer offset (BA00-BA08)
                      51   52      11DB
                                11DD  4062          ASSUME  PTE$S_PFN GE 13
                00   09   50  F0  11DD  4063          INSV    R0,#9,#13,R1                ; Copy BA09-BA21
                             51  11E1
                             50  D4  11E2  4064          CLRL    R0                          ; Use R0 as receive buffer index
;RNG0001        00AC C640    51  D0  11E4     .1 125$:  MOVL    R1,CDB_L_RCV_PA(R6)[R0] ; Save receive physical address
;RNG0001        00C4 C640    52  D0  11EA     .2        MOVL    R2,CDB_L_RCV_VA(R6)[R0] ; Save receive virtual adress
;RNG0001           000005EA  8F  C0  11F0     .3        ADDL    #MAX_BUFSIZ_UV1,R1          ; Skip to next buffer
                             51  11F6
;RNG0001           000005EA  8F  C0  11F7     .4        ADDL    #MAX_BUFSIZ_UV1,R2          ;       ...
                             52  11FD
;RNG0001            E2   50  05  F2  11FE     .5        AOBLSS  #MAX_C_RCVUV1,R0,125$       ; Loop if more
;RNG0001                     50  D4  1202     .6        CLRL    R0                          ; Use R0 as transmit buffer index
;RNG0001        00C0 C640    51  D0  1204     .7 130$:  MOVL    R1,CDB_L_XMT_PA(R6)[R0] ; Save transmit physical address
;RNG0001        00D8 C640    52  D0  120A     .8        MOVL    R2,CDB_L_XMT_VA(R6)[R0] ; Save transmit virtual address
;RNG0001           000005EA  8F  C0  1210     .9        ADDL    #MAX_BUFSIZ_UV1,R1          ; Skip to next buffer
                             51  1216
;RNG0001           000005EA  8F  C0  1217    .10        ADDL    #MAX_BUFSIZ_UV1,R2          ;       ...
                             52  121D
;RNG0001            E2   50  01  F2  121E    .11        AOBLSS  #MAX_C_XMTUV1,R0,130$       ; Loop if more
-11                                 1222  4076
                      54   56  D0  1222  4077 140$:    MOVL    R6,R4                       ; Set R4 to CDB address
                                1225  4078          POPQ    R6                          ; Restore R6, R7
                                1228  4079          ;
                                1228  4080          ; Setup fork process to start CDB timer
                                1228  4081          ;
                             38  BB  1228  4082          PUSHR   #^M<R3,R4,R5>               ; Save registers
                             03  E2  122A  4083          BBSS    #CDB_STS_V_TIMER,-          ; Br if timer already going
                   24 0246 C4        122C  4084                  CDB_B_STS(R4),150$          ;
                             04  88  1230  4085          BISB    #DPT$M_NOUNLOAD,-           ; Do not allow driver to be unloaded
                 0000000D'EF         1232  4086                  DPT$TAB+DPT$B_FLAGS         ;  while the TQE is active
                55   0214 C4  DE  1237  4087          MOVAL   CDB_L_TQE(R4),R5            ; Get the TQE address
                             7D  123C  4088          MOVQ    #TQE_DELTA,TQE$Q_DELTA(R5)  ; Set the delta time
             00 01312D00 8F      123D
                   20 A5   000000  1242
                53   1E2F'CF  9E  1248  4089          MOVAB   W^TQE_TIMER,R3              ; Set address of timer wakeup routine
                  2C A5   05  90  124D  4090          MOVB    #TQE$C_SSREPT,TQE$L_RQPID(R5) ; Set the TQE request type
                       00CD   30  1251  4091          BSBW    FORK_TIMER                  ; Create fork process for TQE
                                1254  4092          ;
                                1254  4093          ; Get hardware CSR address
                                1254  4094          ;
                             38  BA  1254  4095 150$:    POPR    #^M<R3,R4,R5>               ; Restore registers
;RNG0001         52   0114 C4  D0  1256     .1        MOVL    CDB_L_CSR(R4),R2            ; Get CSR adddress
-2                                  125B  4098          ;
                                125B  4099          ; Master reset device
                                125B  4100          ;
                                125B  4101          DSBINT  UCB$B_DIPL(R5)              ; Sync access to UCB
                  0E A2   02  B0  1262  4102          MOVW    #XQ_CSR_M_RESET,CSR(R2)     ; Master Reset device
```

```
                                1266  4103 ;
                                1266  4104 ; The master reset will take some time to complete ...
                                1266  4105 ; so we will delay to give the QNA some time.
                                1266  4106 ;
:RNG0001                        1266    .1          TIMEWAIT #1,#XQ_CSR_M_ERR,CSR(R2),W ; Wait for 10 usec, bit should not
:RNG0001                        128C    .2                                              ;   ..set
:RNG0001          51   24 A5    D0  128C    .3          MOVL    UCB$L_CRB(R5),R1        ; Get CRB adddress
:RNG0001          51   2C A1    D0  1290    .4          MOVL    CRB$L_INTD+VEC$L_IDB(R1),R1 ; Get IDB address
:RNG0001          51   0B A1    9A  1294    .5          MOVZBL  IDB$B_VECTOR(R1),R1     ;   Get vector address (divided by 4)
:RNG0001    51    51     02    78  1298    .6          ASHL    #2,R1,R1                ;   Convert to real vector address
:RNG0001      0C A2    51    B0  129C    .7          MOVW    R1,VECTOR(R2)           ; Set vector address
-2                              12A0  4109 ;
                                12A0  4110 ; Copy the Ethernet Hardware Address
                                12A0  4111 ;
                  0250 C4   62  90  12A0  4112          MOVB    PHYADD0(R2),CDB_G_HWA(R4) ; Save Hardware address
                       02 A2    90  12A5  4113          MOVB    PHYADD1(R2),CDB_G_HWA+1(R4) ;   ...
                            0251 C4   12A8
                       04 A2    90  12AB  4114          MOVB    PHYADD2(R2),CDB_G_HWA+2(R4) ;   ...
                            0252 C4   12AE
                       06 A2    90  12B1  4115          MOVB    PHYADD3(R2),CDB_G_HWA+3(R4) ;   ...
                            0253 C4   12B4
                       08 A2    90  12B7  4116          MOVB    PHYADD4(R2),CDB_G_HWA+4(R4) ;   ...
                            0254 C4   12BA
                       0A A2    90  12BD  4117          MOVB    PHYADD5(R2),CDB_G_HWA+5(R4) ;   ...
                            0255 C4   12C0
                                12C3  4118 ;
                                12C3  4119 ; Set CSR mode and enable receiver
                                12C3  4120 ;
                    0E A2   02  AA  12C3  4121          BICW    #XQ_CSR_M_RESET,CSR(R2) ; Clear the master reset
:RNG0001                0100 8F   A8  12C7    .1          BISW    #XQ_CSR_M_ILOOP,CSR(R2) ; DISABLE LOOPBACK
                          0E A2       12CB
:RNG0001                        12CD    .2
:RNG0001                        12CD    .3          ASSUME  NMA$C_LINCN_NOR EQ 0
:RNG0001                        12CD    .4          ASSUME  NMA$C_LINCN_LOO EQ 1
:RNG0001          06 0249 C4    E9  12CD    .5          BLBC    CDB_B_CON(R4),160$      ; Br if LOOPBACK is disabled
:RNG0001              0300 8F   A8  12D2    .6          BISW    #XQ_CSR_M_ILOOP!XQ_CSR_M_ELOOP,- ; Else, set internal extended
:RNG0001                0E A2       12D6    .7                  CSR(R2)                 ;   LOOPBACK
-5                      0040 8F   A8  12D8  4127 160$:  BISW    #XQ_CSR_M_INTENA,CSR(R2); Enable transmit interrupts
                          0E A2       12DC
                          44 A4   B0  12DE  4128          MOVW    CDB_L_RRINGPA(R4),-     ; Set address of receive list entry
                          04 A2       12E1  4129                  RCVLIST(R2)             ;
                          46 A4   90  12E3  4130          MOVB    CDB_L_RRINGPA+2(R4),-   ;   and high order part
                          06 A2       12E6  4131                  RCVLIST1(R2)
                          68 A4   B0  12E8  4132          MOVW    CDB_L_XRINGPA(R4),-     ; Set address of transmit list entry
                          08 A2       12EB  4133                  XMTLIST(R2)             ;
                          6A A4   90  12ED  4134          MOVB    CDB_L_XRINGPA+2(R4),-   ;   and high order part
                          0A A2       12F0  4135                  XMTLIST1(R2)
                                      12F2  4136          ENBINT                          ; Re-enable interrupts
                                      12F5  4137 ;
                                      12F5  4138 ; Initialize QNA mode.
                                      12F5  4139 ;
                           006C   30  12F5  4140          BSBW    SETUP_MODE              ; Setup the QNA mode
                         25 50    E9  12F8  4141          BLBC    R0,180$                 ; Exit if error
                            00    90  12FB  4142          MOVB    #XQ_FC_V_INIT,-         ; Set function request
                         20 A2       12FD  4143                  CXB$B_XQ_FUNC(R2)       ;
                            9E   16  12FF  4144          JSB     @(SP)+                  ; Complete function request
                                      1301  4145          SETBIT  #CDB_STS_V_RUN,-        ; Indicate QNA is running
```

```
                                1301  4146                    CDB_B_STS(R4)                   ;
                                1307  4147            SETBIT  #UCB$V_XQ_RUN,-                 ; Indicate UNIT is running
                                1307  4148                    UCB$W_DEVSTS(R5)                ;
;RNG0001                        130C    .1
;RNG0001                        130C    .2            ASSUME  NMA$C_LINCN_NOR EQ 0
;RNG0001                        130C    .3            ASSUME  NMA$C_LINCN_LOO EQ 1
;RNG0001           09 0249 C4   E8  130C    .4            BLBS    CDB_B_CON(R4),170$      ; Br if LOOPBACK is enabled, don't
;RNG0001                        1311    .5                                                    ;  enable receive interrupts
;RNG0001        50   0114 C4   D0  1311    .6            MOVL    CDB_L_CSR(R4),R0        ; Else, get CSR adddress
;RNG0001          0E A0   01   A8  1316    .7            BISW    #XQ_CSR_M_RCVENA,CSR(R0); Enable receive interrupts
;RNG0001                0116   30  131A    .8 170$:      BSBW    FILLRCVLIST             ; Start the receives
-4                      50   01   9A  131D  4153          MOVZBL  S^#SS$_NORMAL,R0        ; Return success
                                05  1320  4154 180$:      RSB                             ; Return to caller
                                1321  4155
                                1321  4156 ;++
                                1321  4157 ; FORK_TIMER - Routine to create a fork process to start a timer
                                1321  4158 ;
                                1321  4159 ; This routine starts up a FORK process which is used to start a timer.
                                1321  4160 ;
                                1321  4161 ; Inputs:
                                1321  4162 ;
                                1321  4163 ;       R3 = Address of system routine to handle the timer expiration.
                                1321  4164 ;       R5 = Address of TQE block
                                1321  4165 ;
                                1321  4166 ;       IPL = Greater than Queueast IPL
                                1321  4167 ;
                                1321  4168 ; Outputs:
                                1321  4169 ;
                                1321  4170 ;       Fork process is started.
                                1321  4171 ;       R3, R4 are destroyed by EXE$FORK
                                1321  4172 ;
                                1321  4173 ;--
                                1321  4174 FORK_TIMER:                                       ; Create fork process to start timer
                   0A A5   0F   90  1321  4175          MOVB    #DYN$C_TQE,FKB$B_TYPE(R5) ; Set structure type
                   0B A5   06   90  1325  4176          MOVB    #IPL$_QUEUEAST,FKB$B_FIPL(R5) ; Set IPL of fork process
                        32'AF   9F  1329  4177          PUSHAB  B^START_TIMER             ; Push address of fork process
                00C00000'GF   17  132C  4178          JMP     G^EXE$FORK                ; Create fork process to start timer
                                1332  4179
                                1332  4180 ;++
                                1332  4181 ; START_TIMER - Fork process to start the CDB timer
                                1332  4182 ;
                                1332  4183 ; This routine starts up the CDB which is used to monitor the QNA controller
                                1332  4184 ; for proper operation.
                                1332  4185 ;
                                1332  4186 ; Inputs:
                                1332  4187 ;
                                1332  4188 ;       R5 = Address of TQE block
                                1332  4189 ;
                                1332  4190 ;       IPL = Queueast IPL
                                1332  4191 ;
                                1332  4192 ; Implicit inputs:
                                1332  4193 ;
                                1332  4194 ;       TQE$Q_DELTA(R5) = Delta time interval
                                1332  4195 ;       TQE$L_RQPID(R5) = TQE request type (SSSNGL or SSREPT)
                                1332  4196 ;
                                1332  4197 ; Outputs:
                                1332  4198 ;
```

```
                     1332  4199 ;       R0-R3 are destroyed.
                     1332  4200 ;       TQE element added to timer queue
                     1332  4201 ;
                     1332  4202 ;--
                     1332  4203 START_TIMER:
                     1332  4204         DSBINT  #IPL$_TIMER                 ; Raise IPL
0C A5    10 A5   D0  1338  4205         MOVL    FKB$L_FR3(R5),TQE$L_FPC(R5) ; Set address of timer wakeup
         2C A5   90  133D  4206         MOVB    TQE$L_RQPID(R5),-           ; Set TQE request type
         0B A5       1340  4207                 TQE$B_RQTYPE(R5)            ;
   50    20 A5   7D  1342  4208         MOVQ    TQE$Q_DELTA(R5),R0          ; Get delta time
00000000'GF   C0  1346  4209         ADDL    G^EXE$GQ_SYSTIME,R0         ; Add in current time
         50      134C
00000004'GF   D8  134D  4210         ADWC    G^EXE$GQ_SYSTIME+4,R1       ;       ...
         51      1353
00000000'GF   16  1354  4211         JSB     G^EXE$INSTIMQ               ; Insert element on timer queue
                  135A  4212         ENBINT                              ; Restore IPL
                  135D  4213         RSB
```

```
                     135E  4215          .SBTTL  SETUP_MODE - SETUP THE TRANSMIT BUFFER TO INIT QNA
                     135E  4216 ;++
                     135E  4217 ; SETUP_MODE - SETUP THE TRANSMIT BUFFER TO INIT QNA
                     135E  4218 ;
                     135E  4219 ; Functional description:
                     135E  4220 ;
                     135E  4221 ; This routine initializes the TRANSMIT buffer the sets up the QNA operating
                     135E  4222 ; mode.
                     135E  4223 ;
                     135E  4224 ; Inputs:
                     135E  4225 ;
                     135E  4226 ;       R3 = IRP address
                     135E  4227 ;       R4 = CDB address
                     135E  4228 ;       R5 = UCB address
                     135E  4229 ;
                     135E  4230 ;       IPL = FIPL
                     135E  4231 ;
                     135E  4232 ; Outputs:
                     135E  4233 ;
                     135E  4234 ;       R0 = status of request
                     135E  4235 ;
                     135E  4236 ;       R1,R2 are destroyed
                     135E  4237 ;--
                     135E  4238 SETUP_ERR:                              ; Setup error
50   0124 8F    3C   135E  4239          MOVZWL  #SS$_INSFMEM,R0          ; Set error return
                05   1363  4240          RSB                              ; Return to caller
                     1364  4241
                     1364  4242 SETUP_MODE:
          53    DD   1364  4243          PUSHL   R3                       ; Save R3
51   00C8 8F    3C   1366  4244          MOVZWL  #CXB$C_HEADER+INIT_C_BUFSIZE,R1 ; Size of init buffer
 00000000'GF    16   136B  4245          JSB     G^EXE$ALONONPAGED        ; Allocate the SETUP Transmit buffer
          53 8ED0   1371  4246          POPL    R3                       ; Restore R3
       E7 50    E9   1374  4247          BLBC    R0,SETUP_ERR             ; Exit if error
          3C    BB   1377  4248          PUSHR   #^M<R2,R3,R4,R5>         ; Save registers
    62    00    2C   1379  4249          MOVC5   #0,(R2),#-1,R1,(R2)      ; Fill structure with BROADCAST!
 51    FF 8F        137C
          62        137F
          3C    BA   1380  4250          POPR    #^M<R2,R3,R4,R5>         ; Restore registers
                     1382  4251          ;
                     1382  4252          ; Initialize buffer to look like a CXB
                     1382  4253          ;
0094 C3   52    D0   1382  4254          MOVL    R2,IRP$L_XQ_SETUP(R5)    ; Save address of setup buffer
                     1387  4255          ASSUME  CXB$B_TYPE EQ CXB$W_SIZE+2
                     1387  4256          ASSUME  CXB$B_CODE EQ CXB$B_TYPE+1
                D0   1387  4257          MOVL    #<DYN$C_CXB@16>!-        ; Set size and type of structure
                     1388  4258                  CXB$C_HEADER+INIT_C_BUFSIZE,-
 001B00C8 8F        1388  4259                  CXB$W_SIZE(R2)
       08 A2        138D
    0080 8F    B0   138F  4260          MOVW    #INIT_C_BUFSIZE,CXB$W_BCNT(R2) ; Set size of transfer
      1A A2        1393
                     1395  4261          ;
                     1395  4262          ; Initialize QNA mode word
                     1395  4263          ;
                     1395  4264          ASSUME  NMA$C_STATE_ON EQ 0
                     1395  4265          ASSUME  NMA$C_STATE_OFF EQ 1
    0244 C4    B4   1395  4266          CLRW    CDB_W_MODE(R4)           ; Init mode word
 05 0247 C4    E8   1399  4267          BLBS    CDB_B_PRM(R4),10$        ; Br if promiscuous state is OFF
```

```
                02    A8  139E  4268          BISW    #CDB_MOD_M_PROM,-       ; Else, enable promiscuous mode
          0244  C4        13A0  4269                  CDB_W_MODE(R4)          ;
     05   0248  C4    E8  13A3  4270 10$:     BLBS    CDB_B_MLT(R4),20$       ; Br if multicast state is OFF
                01    A8  13A8  4271          BISW    #CDB_MOD_M_MULTI,-      ; Else, enable all MULTICASTS
          0244  C4        13AA  4272                  CDB_W_MODE(R4)          ;
          0244  C4    A8  13AD  4273 20$:     BISW    CDB_W_MODE(R4),CXB$W_BCNT(R2) ; Set mode bits
            1A  A2        13B1
     18   A2    3A    B0  13B3  4274          MOVW    #CXB$T_T_DATA,CXB$W_BOFF(R2) ; Set offset to start of data
     24   A2    53    D0  13B7  4275          MOVL    R3,CXB$L_T_IRP(R2)      ; Save IRP address in CXB
                        13BB  4276          ;
                        13BB  4277          ; Initialize physical address
                        13BB  4278          ;
     50   024A  C4    9E  13BB  4279          MOVAB   CDB_G_PHA(R4),R0        ; Point to Physical Address
          51    3B    A2  9E  13C0  4280      MOVAB   CXB$T_T_DATA+1(R2),R1   ; Point to setup buffer (skip 1st col)
                61    80  90  13C4  4281      MOVB    (R0)+,(R1)              ; Stuff the physical address
          08    A1    80  90  13C7  4282      MOVB    (R0)+,8(R1)             ;       ...
          10    A1    80  90  13CB  4283      MOVB    (R0)+,16(R1)            ;       ...
          18    A1    80  90  13CF  4284      MOVB    (R0)+,24(R1)            ;       ...
          20    A1    80  90  13D3  4285      MOVB    (R0)+,32(R1)            ;       ...
          28    A1    80  90  13D7  4286      MOVB    (R0)+,40(R1)            ;       ...
                        13DB  4287          ;
                        13DB  4288          ; Initialize multicast addresses
                        13DB  4289          ;
                        13DB  4290          PUSHQ   R2                      ; Save setup buffer, IRP address
     52   025C  C4    9A  13DE  4291          MOVZBL  CDB_B_MULTI(R4),R2      ; Get number of multicast addresses
                38    13  13E3  4292          BEQL    80$                     ; Br if none
                52    0C  91  13E5  4293      CMPB    #12,R2                  ; Is count okay?
                04    1E  13E8  4294          BGEQU   30$                     ; Br if yes
                        13EA  4295          BUG_CHECK NOBUFPCKT,FATAL       ; Else, error
                        13EE  4296
                53    06  9A  13EE  4297 30$:     MOVZBL  #6,R3               ; Only 6 slots left in first half of
                        13F1  4298                                          ;  setup buffer
     50   025E  C4    9E  13F1  4299          MOVAB   CDB_G_MULTI(R4),R0      ; Point to multicast address list
                09    11  13F6  4300          BRB     50$                     ; Start with first half
                        13F8  4301          ;
                        13F8  4302          ; Check if first half of setup buffer is full
                        13F8  4303          ;
                  06    53  F5  13F8  4304 40$:     SOBGTR  R3,50$              ; Br if first half of buffer still open
                51    39  C0  13FB  4305      ADDL    #64-7,R1                ; Skip to second half of buffer
                53    06  9A  13FE  4306      MOVZBL  #6,R3                   ; Reset count for second half
                        1401  4307                                          ; Leave last address as BROADCAST
                        1401  4308          ;
                        1401  4309          ; Store multicast addresses
                        1401  4310          ;
                51    D6  1401  4311 50$:     INCL    R1                      ; Skip to next column
                61    80  90  1403  4312      MOVB    (R0)+,(R1)              ; Store multicast address
          08    A1    80  90  1406  4313      MOVB    (R0)+,8(R1)             ;       ...
          10    A1    80  90  140A  4314      MOVB    (R0)+,16(R1)            ;       ...
          18    A1    80  90  140E  4315      MOVB    (R0)+,24(R1)            ;       ...
          20    A1    80  90  1412  4316      MOVB    (R0)+,32(R1)            ;       ...
          28    A1    80  90  1416  4317      MOVB    (R0)+,40(R1)            ;       ...
                DB    52  F5  141A  4318      SOBGTR  R2,40$                  ; Br if more
                        141D  4319
                        141D  4320 80$:     POPQ    R2                      ; Restore setup buffer, IRP address
                50    01  9A  1420  4321      MOVZBL  #SS$_NORMAL,R0          ; Return success
                9E    16  1423  4322      JSB     @(SP)+                  ; Call back caller as co-routine
     00E0 D4    62    0E  1425  4323      INSQUE  (R2),@CDB_Q_XMTREQ+4(R4) ; Insert request on xmit queue
```

```
                     142A  4324          SETBIT  #CDB_STS_V_SETUP,-      ; Indicate that SETUP is in progress
                     142A  4325                  CDB_B_STS(R4)           ;
F054    31           1430  4326          BRW     XMT_ACT_START           ; Startup the XMIT process
                     1433  4327
```

```
                     1433  4329          .SBTTL  FILLRCVLIST - FILL RECEIVE BUFFER LIST
                     1433  4330 ;++
                     1433  4331 ; FILLRCVLIST - FILL RECEIVE BUFFER LIST
                     1433  4332 ;
                     1433  4333 ; Functional description:
                     1433  4334 ;
                     1433  4335 ; This routine fills the receive buffer list up to the quota allocated
                     1433  4336 ; at unit initialization.  It also gives any receive buffers allocated
                     1433  4337 ; to the receiver.
                     1433  4338 ;
                     1433  4339 ; Inputs:
                     1433  4340 ;
                     1433  4341 ;       R2 = Buffer Address (ADDRCVLIST ONLY)
                     1433  4342 ;       R4 = CDB address
                     1433  4343 ;
                     1433  4344 ;       IPL = FIPL
                     1433  4345 ;
                     1433  4346 ; Outputs:
                     1433  4347 ;
                     1433  4348 ;       R0-R2 is destroyed.
                     1433  4349 ;       All other registers are preserved.
                     1433  4350 ;
                     1433  4351 ;--
                     1433  4352          .ENABL  LSB
                     1433  4353 FILLRCVLIST::                          ; Fill receive buffer list
            52   D4  1433  4354          CLRL    R2                      ; No buffer here
                     1435  4355
                     1435  4356 ADDRCVLIST::                          ; Add a buffer to the receive list
                  38 BB    1435  4357          PUSHR   #^M<R3,R4,R5>           ; Save registers
            55  0110 C4  D0  1437  4358          MOVL    CDB_L_UCB0(R4),R5       ; Get UCB address of unit #0
;RNG0001               01  E1  143C    .1          BBC     #CDB_STS_V_RUN,-        ; Br if device not running
;RNG0001          5D 0246 C4      143E    .2                  CDB_B_STS(R4),40$
;RNG0001             0108 C4  B1  1442    .3 10$:     CMPW    CDB_W_BSZ(R4),-         ; Can new block be allocated ?
;RNG0001             010A C4      1446    .4                  CDB_W_QUOTA(R4)
;RNG0001                  3B  1A  1449    .5          BGTRU   35$                     ; If GTRU then no, stop loop
;RNG0001                  52  D5  144B    .6          TSTL    R2                      ; Buffer already allocated?
;RNG0001                  17  12  144D    .7          BNEQ    20$                     ; Br if so
;RNG0001                  51  D4  144F    .8          CLRL    R1                      ; Zero size
;RNG0001                      A1  1451    .9          ADDW3   #CXB$C_HEADER+-         ; Determine block size needed
;RNG0001                      1452  .10                      CXB$C_TRAILER,-         ;
;RNG0001             004C 8F      1452  .11                  CDB_W_BSZ(R4),R1        ;
            51   0108 C4          1455
;RNG0001      00000000'GF  16  1459  .12          JSB     G^EXE$ALONONPAGED       ; Allocate the memory
;RNG0001            1C 50  E9  145F  .13          BLBC    R0,30$                  ; If failure then done
;RNG0001         08 A2 51  B0  1462  .14          MOVW    R1,CXB$W_SIZE(R2)       ; Insert size
;RNG0001             0108 C4  A2  1466  .15 20$:     SUBW    CDB_W_BSZ(R4),-         ; Subtract from quota
;RNG0001             010A C4      146A  .16                  CDB_W_QUOTA(R4)         ;
-16                      1B  90  146D  4375 25$:     MOVB    S^#DYN$C_CXB,-          ; Insert type
                      0A A2      146F  4376                  CXB$B_TYPE(R2)          ;
                   20 A2  02  90  1471  4377          MOVB    #XQ_FC_V_RECV,CXB$B_XQ_FUNC(R2) ; Set function request
                00EC C4   62  0E  1475  4378          INSQUE  (R2),CDB_Q_RCVBUF(R4)   ; Insert block on list
                          52  D4  147A  4379          CLRL    R2                      ; No more buffers given
                          C4  11  147C  4380          BRB     10$                     ; Continue
                              147E  4381 ;
                              147E  4382 ; Buffer allocation failure
                              147E  4383 ;
                              147E  4384 30$:     SETBIT  #XM$V_STS_BUFFAIL,-     ; Set buffer alloc failure
```

```
                          147E  4385                    CDB_L_DEVDEPEND(R4)     ;
                    24    11  1484  4386              BRB     50$               ; And give any receives to device
                          1486  4387
                          1486  4388 35$:           CLRBIT  #XM$V_STS_BUFFAIL,-   ; Clear buffer alloc failure
                          1486  4389                    CDB_L_DEVDEPEND(R4)     ;
                    52    D5  148C  4390              TSTL    R2                ; Any buffer?
                    1A    13  148E  4391              BEQL    50$               ; Br if not
                02EE C4   91  1490  4392              CMPB    CDB_B_AQUOTA(R4),-  ; Can we use the additional quota?
                02EF C4       1494  4393                    CDB_B_MQUOTA(R4)    ;
                    06    1E  1497  4394              BGEQU   40$               ; Br if not
:RNG0001        02EE C4   96  1499    .1              INCB    CDB_B_AQUOTA(R4)  ; Else, increment the additional quota
-1                  CE    11  149D  4396              BRB     25$               ; Use buffer, but don't let
                          149F  4397                                            ;  QUOTA go negative
                          149F  4398
                    50    52  D0  149F  4399 40$:     MOVL    R2,R0             ; Get address of buffer
                    06    13  14A2  4400              BEQL    50$               ; Br if none
            00000000'GF   16  14A4  4401              JSB     G^COM$DRVDEALMEM  ; Deallocate buffer
                          14AA  4402
                    03    10  14AA  4403 50$:        BSBB    START_RECEIVE     ; Start the receives
                    38    BA  14AC  4404              POPR    #^M<R3,R4,R5>     ; Restore registers
                    05    14  14AE  4405              RSB
                          14AF  4406              .DSABL  LSB
```

```
                        14AF  4408          .SBTTL  START_RECEIVE - START ANY RECEIVE REQUESTS PENDING
                        14AF  4409 ;++
                        14AF  4410 ; START_RECEIVE - START ANY RECEIVE REQUESTS
                        14AF  4411 ;
                        14AF  4412 ; Functional description:
                        14AF  4413 ;
                        14AF  4414 ; This routine attemptes to start any receives that may be pending.  This
                        14AF  4415 ; involves dequeueing a free receive buffer, mapping it, and loading it's
                        14AF  4416 ; address and size into the device.
                        14AF  4417 ;
                        14AF  4418 ; Inputs:
                        14AF  4419 ;
                        14AF  4420 ;       R4 = CDB address
                        14AF  4421 ;       R5 = UCB address of unit # 0
                        14AF  4422 ;
                        14AF  4423 ;       IPL = FIPL
                        14AF  4424 ;
                        14AF  4425 ; Outputs:
                        14AF  4426 ;
                        14AF  4427 ;       R0-R3 are destroyed.
                        14AF  4428 ;       All other registers are preserved.
                        14AF  4429 ;
                        14AF  4430 ;--
                        14AF  4431
                        14AF  4432 START_RECEIVE::                              ; Start receive operation
                        14AF  4433          PUSHQ   R6                          ; Save R6, R7
                  01    E1    14B2  4434          BBC     #CDB_STS_V_RUN,-        ; Br if device is not running
              2B 0246 C4        14B4  4435                  CDB_B_STS(R4),10$    ;
                        14B8  4436          ;
;RNG0001                 14B8    .1          ; For MicroVAX I, we will not use map registers.
;RNG0001                 14B8    .2          ;
;RNG0001                 14B8    .3          CPUDISP <<790,5$>,-
;RNG0001                 14B8    .4                   <780,5$>,-
;RNG0001                 14B8    .5                   <750,5$>,-
;RNG0001                 14B8    .6                   <730,5$>,-
;RNG0001                 14B8    .7                   <UV2,5$>,-
;RNG0001                 14B8    .8                   <UV1,40$>>             ; For MicroVAX I, use alternate path
;RNG0001                 14D4    .9
;RNG0001                 14D4    .10 5$:     ;  All but MicroVAX I
;RNG0001                 14D4    .11
;RNG0001           07    00    EB  14D4    .12         FFC     #0,#MAX_C_RCV-1,CDB_B_RCVMAP(R4),R7 ; Get a free mapping slot
                  57    1A    A4  14D7
-9                      07    13  14DA  4446          BEQL    10$                 ; Br if none - just exit
                53    00EC  D4    0F  14DC  4447          REMQUE  @CDB_Q_RCVBUF(R4),R3 ; Get a free buffer
                        04    1C  14E1  4448          BVC     20$                 ; Br if buffer found
                        14E3  4449 10$:   POPQ    R6                          ; Restore R6, R7
                        05    14E6  4450          RSB                         ; Return to caller
                        14E7  4451 ;
                        14E7  4452 ; Mark slot in use and create buffer address/character count image
                        14E7  4453 ; in receive buffer and load UNIBUS adapter map registers.
                        14E7  4454 ;
                        14E7  4455 20$:   SETBIT  R7,CDB_B_RCVMAP(R4)         ; Mark slot in use
                  22 A3 57    90  14EC  4456          MOVB    R7,CXB$B_XQ_SLOT(R3) ; Save mapping slot index
                  52    24 A5 D0  14F0  4457          MOVL    UCB$L_CRB(R5),R2    ; Get CRB address
                        14F4  4458 ;
                        14F4  4459 ; Find next ring entry and insert data
                        14F4  4460 ;
```

```
                     56    19 A4    9A  14F4  4461          MOVZBL  CDB_B_NEXTRCV(R4),R6     ; Get next ring entry
                           19 A4    96  14F8  4462          INCB    CDB_B_NEXTRCV(R4)       ; Bump ring pointer
                           F8 8F    8A  14FB  4463          BICB    #^C<MAX_C_RCV-1>,-      ; Modulo rcv ring size
                           19 A4        14FE  4464                    CDB_B_NEXTRCV(R4)     ;
                   23 A3     56    90  1500  4465          MOVB    R6,CXB$B_XQ_RING(R3)    ; Save ring entry number
                56    7C A446     D0  1504  4466          MOVL    CDB_L_RRINGVA(R4)[R6],R6; Get virtual address of ring entry
                            66     B4  1509  4467          CLRW    RCV_W_FLAG(R6)          ; Zero the FLAG word
                         8000 8F    B0  150B  4468          MOVW    #RCV_STS_M_LAST,-       ; Init the STATUS word
                           08 A6        150F  4469                  RCV_W_STS(R6)         ;
                                        1511  4470          PUSHQ   R3                      ; Save IRP, CDB address
                                        1514  4471 ;
                                        1514  4472 ; The QNA's receive buffer size must be a multiple of 2
                                        1514  4473 ;
;RNG0001                              50    D4  1514    .1          CLRL    R0                      ; Zero out high part of R0 longword
                                             1516  4474          ASSUME  <XQ_C_HEADER+XQ_C_CNTSIZ & 1 > EQ 0
                                     10    A1  1516  4475          ADDW3   #XQ_C_HEADER+XQ_C_CNTSIZ,- ; Calculate message length
                         50    0108 C4        1518  4476                  CDB_W_BSZ(R4),R0        ;                  ...
                           50    FF 8F    78  151C  4477          ASHL    #-1,R0,R0               ; Convert byte count to WORD count
                                     50         1520
                             06 A6    50    AE  1521  4478          MNEGW   R0,RCV_W_LEN(R6)        ; Store length (2's complement)
                             51    38 A3    9E  1525  4479          MOVAB   CXB$T_R_DATA(R3),R1     ; Get receive buffer data addr
                             04 A6    51    B0  1529  4480          MOVW    R1,RCV_W_ADDR(R6)       ; Set BA0-BA8
                        57    1C A447    DE  152D  4481          MOVAL   CDB_L_RCVMAP(R4)[R7],R7 ; Get mapping info slot address
                        07    09    67    F0  1532  4482          INSV    (R7),#9,#7,RCV_W_ADDR(R6) ; Set BA9-BA15 from map reg
                                 04 A6         1536
                        67    06    07    EF  1538  4483          EXTZV   #7,#6,(R7),R0           ; Get BA16-BA21 also
                                     50         153C
                             02 A6    50    B0  153D  4484          MOVW    R0,RCV_W_ADDRHI(R6)     ; Set BA16-BA21 & zero descriptor bits
                             0A A6    01    B0  1541  4485          MOVW    #1,RCV_W_LENB(R6)       ; Set low byte not equal to high byte
                                      54    D4  1545  4486          CLRL    R4                      ; Use direct data path for rcvs
                                             1547  4487          ASSUME  VEC$W_NUMREG EQ VEC$W_MAPREG+2
                                52    02 A7    9A  1547  4488          MOVZBL  2(R7),R2                ; Set number of map registers
                                     53    67    3C  154B  4489          MOVZWL  (R7),R3                 ; Set first map register number
                             00000000'GF    16  154E  4490          JSB     G^IOC$LOADUBAMAPN       ; Load the map registers
                                             1554  4491          POPQ    R3                      ; Restore IRP, CDB address
                                             155?  4492 ;
                                             1557  4493 ; Disable interrupts and queue request to input queue
                                             1557  4494 ;
;RNG0001                                  8000 8F    A8  1557    .1          BISW    #RCV_DSC_M_VALID,-      ; Set descriptor bits
;RNG0001                                    02 A6        155B    .2                  XMT_W_ADDRHI(R6)        ;  "VALID" buffer address
;RNG0001                                  0106 C4    96  155D    .3          INCB    CDB_B_RCVCNT(R4)        ; Tally one more receive in progress
;RNG0001                                                 1561    .4
;RNG0001                                     0074    30  1561   .10          BSBW    LOAD_RECV               ; Give (new) ring address to QNA
;RNG0001                                     FF6D    31  1564   .12          BRW     5$                      ; Let's try it again
;RNG0001C1                                           1567   .13
;RNG0001                                             1567   .14
;RNG0001                                             1567   .15 40$:    ;**********************
;RNG0001                                             1567   .16         ; For MicroVAX I, only.
;RNG0001                                             1567   .17         ;**********************
-15
                                                     1567  4510
                                                     1567  4511          ASSUME  MAX_C_RCV LE 8
                                                     1567  4512          ASSUME  MAX_C_RCVUV1 LT MAX_C_RCV
                                     05    00    EB  1567  4513          FFC     #0,#MAX_C_RCVUV1,CDB_B_RCVMAP(R4),R7 ; Get a free mapping slot
                                57    1A A4        156A
                                             07    13  156D  4514          BEQL    50$                     ; Br if none - just exit
                                53    00EC D4    0F  156F  4515          REMQUE  @CDB_Q_RCVBUF(R4),R3    ; Get a free buffer
                                             04    1C  1574  4516          BVC     60$                     ; Br if buffer found
```

```
                          1576  4517 50$:    POPQ    R6                          ; Restore R6, R7
                    05    1579  4518         RSB                                 ; Return to caller
                          157A  4519 ;
                          157A  4520 ; Mark slot in use and create buffer address/character count image
                          157A  4521 ; in receive buffer and load UNIBUS adapter map registers.
                          157A  4522 ;
                          157A  4523 60$:    SETB'T  R7,CDB_B_RCVMAP(R4)         ; Mark slot in use
                 22 A3  57  90    157F  4524         MOVB    R7,CXB$B_XQ_SLOT(R3)    ; Save mapping slot index
                          1583  4525 ;
                          1583  4526 ; Find next ring entry and insert data
                          1583  4527 ;
                 56  19 A4  9A    1583  4528         MOV7BL  CDB_B_NEXTRCV(R4),R6    ; Get next ring entry
                    19 A4  96    1587  4529         INCB    CDB_B_NEXTRCV(R4)       ; Bump ring pointer
                    F8 8F  8A    158A  4530         BICB    #^C<MAX_C_RCV-1>,-      ; Modulo rcv ring size
                    19 A4        158D  4531                  CDB_B_NEXTRCV(R4)      ;
                 23 A3  56  90    158F  4532         MOVB    R6,CXB$B_XQ_RING(R3)    ; Save ring entry number
              56 7C A446  D0    1593  4533         MOVL    CDB_L_RRINGVA(R4)[R6],R6; Get virtual address of ring entry
                    66  B4    1598  4534         CLRW    RCV_W_FLAG(R6)          ; Zero the FLAG word
                  8000 8F  B0    159A  4535         MOVW    #RCV_STS_M_LAST,-       ; Init the STATUS word
                    08 A6        159E  4536                 RCV_W_STS(R6)          ;
                          15A0  4537 ;
                          15A0  4538 ; The QNA's receive buffer size must be a multiple of 2
                          15A0  4539 ;
;RNG0001                   50  D4    15A0    .1         CLRL    R0                  ; Zero out high part of R0 longword
                          15A2  4540         ASSUME  <XQ_C_HEADER+XQ_C_CNTSIZ & 1 > EQ 0
                    10  A1    15A2  4541         ADDW3   #XQ_C_HEADER+XQ_C_CNTSIZ,- ; Calculate message length
                50  0108 C4        15A4  4542                 CDB_W_BSZ(R4),R0       ;
                  50  FF 8F  78    15A8  4543         ASHL    #-1,R0,R0               ; Convert byte count to WORD count
                    50           15AC
                 06 A6  50  AE    15AD  4544         MNEGW   R0,RCV_W_LEN(R6)        ; Store length (2's complement)
              50  00AC C447  D0    15B1  4545         MOVL    CDB_L_RCV_PA(R4)[R7],R0 ; Get receive buffer physical address
                 04 A6  50  B0    15B7  4546         MOVW    R0,RCV_W_ADDR(R6)       ; Set BA00-BA15
                  50  F0 8F  78    15BB  4547         ASHL    #-16,R0,R0              ; Shift down high byte of address
                    50           15BF
                 02 A6  50  B0    15C0  4548         MOVW    R0,RCV_W_ADDRHI(R6)     ; Set BA16-BA21 & zero descriptor bits
                 0A A6  01  B0    15C4  4549         MOVW    #1,RCV_W_LENB(R6)       ; Set low byte not equal to high byte
                          15C8  4550 ;
                          15C8  4551 ; Disable interrupts and queue request to input queue
                          15C8  4552 ;
;RNG0001                 8000 8F  A8    15C8    .1         BISW    #RCV_DSC_M_V,-,D,-      ; Set descriptor bits
;RNG0001                   02 A6        15CC    .2                 RCV_W_ADDRHI(R6)        ; "VALID" buffer address
;RNG0001                 0106 C4  96    15CE    .3         INCB    CDB_B_RCVCNT(R4)        ; Tally one more receive in progress
;RNG0001                          15D2    .4
;RNG0001                    0003  3C    15D2   .10         BSBW    LOAD_RECV               ; Give (new) ring address to QNA
;RNG00C1                          15D5   .12
;RNG0001                    FF8F  31    15D5   .13 90$:    BRW     40$                     ; Let's try it again
```

```
;RNG0001                                15D8   .35          .SBTTL  LOAD_RECV - LOAD CSR'S WITH RECEIVE RING ADDRESS IF NEEDED
;RNG0001                                15D8   .36 ;++
;RNG0001                                15D8   .37 ; LOAD_RECV - LOAD CSR'S WITH RECEIVE RING ADDRESS IF NEEDED
;RNG0001                                15D8   .38 ;
;RNG0001                                15D8   .39 ; Functional description:
;RNG0001                                15D8   .40 ;
;RNG0001                                15D8   .41 ; This routine loads the receive ring address, if the recei  ring has gone
;RNG0001                                15D8   .42 ; invalid.
;RNG0001                                15D8   .43 ;
;RNG0001                                15D8   .44 ; Inputs:
;RNG0001                                15D8   .45 ;
;RNG0001                                15D8   .46 ;       R3 = CXB address for receive
;RNG0001                                15D8   .47 ;       R4 = CDB address
;RNG0001                                15D8   .48 ;       R5 = UCB address
;RNG0001                                15D8   .49 ;
;RNG0001                                15D8   .50 ;       IPL = FIPL
-1                                      15D8  4578 ;
                                        15D8  4579 ; Outputs:
                                        15D8  4580 ;
                                        15D8  4581 ;       R4,R5 are preserved.
                                        15D8  4582 ;
                                        15D8  4583 ;       R0-R3 may be destroyed.
                                        15D8  4584 ;
                                        15D8  4585 ;--
;RNG0001                                 15D8    .1 LOAD_RECV::                          ; Load port command
;RNG0001          52   0114 C4   D0     15D8    .2          MOVL    CDB_L_CSR(R4),R2    ; Get CSR address
;RNG0001                                 15DD    .3 ;
;RNG0001                                 15DD    .4 ;   If the QNA has invalidated the RECEIVE RING, then we must reset the
;RNG0001                                 15DD    .5 ;   starting address of the ring list to point to the current entry.
;RNG0001                                 15DD    .6 ;
;RNG0001            0E A2    20   B3     15DD   .11          BITW    #XQ_CSR_M_RCVINV,CSR(R2); Is receive ring still valid?
;RNC0001                       16   13     15E1   .12          BEQL    40$                 ; Br if yes
;RNG0001              51   23 A3   9A     15E3   .14          MOVZBL  CXB$B_XQ_RING(R3),R1 ; Else, get ring entry number
;RNG0001          51   44 A441   D0       15E7   .15          MOVL    CDB_L_RRINGPA(R4)[R1],R1; Get the buffer mapping value
;RNG0001            04 A2    51   B0     15EC   .16          MOVW    R1,RCVLIST(R2)      ; Set address of receive list entry
;RNG0001            51   F0 8F   78     15F0   .17          ASHL    #-16,R1,R1          ; Shift down high order address bits
                                51       15F4
;RNG0001            06 A2    51   90     15F5   .18          MOVB    R1,RCVLST1(R2)      ;                ...
;RNG0001          00F8 D4    63   0E     15F9   .19 40$:     INSQUE  (R3),@CDB_Q_RCVPND+4(R4) ; Insert CXB on WAITING queue
-58                                     05   15FE  4644          RSB                     ; Return to caller
                                        15FF  4645
```

```
                                15FF  4647          .SBTTL  QNA_INTR - QNA INTERRUPT SERVICE ROUTINE
                                15FF  4648 ;++
                                15FF  4649 ; QNA_INTR - QNA INTERRUPT SERVICE ROUTINE
                                15FF  4650 ;
                                15FF  4651 ; Functional description:
                                15FF  4652 ;
                                15FF  4653 ; This routine services the interrupts generated by the QNA for completion
                                15FF  4654 ; of requests.
                                15FF  4655 ;
                                15FF  4656 ; Inputs:
                                15FF  4657 ;
                                15FF  4658 ;       00(SP) = ADDRESS OF UNIT IDB ADDRESS
                                15FF  4659 ;
                                15FF  4660 ;       R0,R1,R2,R3,R4,R5 ARE AT 04(SP) TO 1C(SP)
                                15FF  4661 ;
                                15FF  4662 ;
                                15FF  4663 ;       IPL = DIPL
                                15FF  4664 ;
                                15FF  4665 ; Outputs:
                                15FF  4666 ;
                                15FF  4667 ;       THE INTERRUPT IS DISMISSED
                                15FF  4668 ;
                                15FF  4669 ; IMPLICIT OUTPUTS:
                                15FF  4670 ;
                                15FF  4671 ;       A Fork process is started to check ring entries.
                                15FF  4672 ;--
                                15FF  4673 QNA_INTR::                               ; DEQNA done interrupt
          54    9E    D0        15FF  4674          MOVL    @(SP)+,R4             ; Get IDB address
     55    18 A4    D0        1602  4675          MOVL    IDB$L_UCBLST(R4),R5   ; Get first UCB address
                                1606  4676          ASSUME  IDB$L_CSR EQ 0
          52    64    D0        1606  4677          MOVL    (R4),R2               ; Get CSR address
;RNG0001        54    00C8 C5   D0    1609    .1          MOVL    UCB$L_XQ_CDB(R5),R4   ; Get CDB address
;RNG0001                2A    13    160E    .2          BEQL    INTEXIT               ; Br if CDB not allocated
;RNG0001                                1610    .3          ASSUME  CDB_STS_V_INITED  EQ  0
;RNG0001          25 0246 C4    E9    1610    .4          BLBC    CDB_B_STS(R4),INTEXIT ; Br if NOT inited
-5                      04    E0    1615  4683          BBS     #CDB_STS_V_ERR,-      ; Br if there was an error
                  1F 0246 C4          1617  4684                  CDB_B_STS(R4),INTEXIT ;
                  53    0E A2   3C    161B  4685          MOVZWL  CSR(R2),R3            ; Fetch the CSR contents
                                161F  4686 ;
                                161F  4687 ; The interrupt bits can only be cleared by writing one's into them, therefore
                                161F  4688 ; we will write one's into all bits which already have one's.
                                161F  4689 ;
                    0E A2   53  B0    161F  4690          MOVW    R3,CSR(R2)            ; Release interrupt interlocks
                53    8080 8F   B3    1623  4691          BITW    #XQ_CSR_M_XMTINT!-    ; Is this a valid interrupt?
                                1628  4692                    XQ_CSR_M_RCVINT,R3  ;
                        04    13    1628  4693          BEQL    20$                   ; Br if no, indicate error
                                162A  4694 ;
                                162A  4695 ; We will now check for any errors.
                                162A  4696 ;
                  0A 53   02    E1    162A  4697 10$:       BBC     #XQ_CSR_V_NXM,R3,30$  ; Br if no errors
                                162E  4698 20$:       SETBIT  #XQ_CSR_V_ERR,R3      ; Set fatal error indicator
                                1632  4699          SETBIT  #CDB_STS_V_ERR,-      ; Ignore futher interrupts
                                1632  4700                  CDB_B_STS(R4)         ;
                        0C    10    1638  4701 30$:       BSBB    SCHED_FORK            ; Schedule a fork process
                                163A  4702
                                163A  4703 INTEXIT:                                 ; Exit interrupt
          50    8E    7D        163A  4704          MOVQ    (SP)+,R0              ; Restore Regs
```

```
                52   8E   7D   163D  4705          MOVQ    (SP)+,R2
                54   8E   7D   1640  4706          MOVQ    (SP)+,R4
                          02   1643  4707          REI                             ; Dismiss the interrupt
```

```
                                1644  4709          .SBTTL  SCHED_FORK - SCHEDULE THE FORK PROCESS
                                1644  4710          .SBTTL  SCHED_FORKC - SCHEDULE THE FORK PROCESS WITH R3 CLEAR
                                1644  4711 ;++
                                1644  4712 ; SCHED_FORK - Schedule the fork process
                                1644  4713 ; SCHED_FORKC - Schedule the fork process with R3 clear
                                1644  4714 ;
                                1644  4715 ; Functional description:
                                1644  4716 ;
                                1644  4717 ; This routine is called to schedule the error and I/O completion fork process.
                                1644  4718 ; The last controller CSR values are saved for examination.  If the
                                1644  4719 ; fork process is already pending, only the last CSR values are saved if there
                                1644  4720 ; was an error.
                                1644  4721 ;
                                1644  4722 ; Inputs:
                                1644  4723 ;
                                1644  4724 ;       R3 = Last CSR value
                                1644  4725 ;       R4 = CDB address
                                1644  4726 ;
                                1644  4727 ;       IPL = DIPL or higher
                                1644  4728 ;
                                1644  4729 ; Outputs:
                                1644  4730 ;
                                1644  4731 ;       R3 is cleared if SCHED_FORKC entry.
                                1644  4732 ;       R4 is destroyed.
                                1644  4733 ;       R5 = CDB address
                                1644  4734 ;
                                1644  4735 ; If XQ_CSR_V_ERR is set in CSR, then the following is returned:
                                1644  4736 ;
;RNG0001                         1644     .1 ;       CDB_L_LASTCSR(R4) = new CSR contents
-1                              1644  4738 ;
                                1644  4739 ;--
                                1644  4740
                                1644  4741 SCHED_FORKC::                                ; Schedule fork process, clr R3
                          53    D4  1644  4742          CLRL    R3                      ; No device error
                                1646  4743 SCHED_FORK::                                ; Schedule fork process
                    0246 C4     02  E2  1646  4744          BBSS    #CDB_STS_V_FORK_PEND,CDB_B_STS(R4),10$ ; Br if fork pending
                                    0C  1648
                                164C  4745          ASSUME  CDB_L_FQFL EQ 0
                          55    54  D0  164C  4746          MOVL    R4,R5                   ; Get CDB fork block address
                             63'AF    9F  164F  4747          PUSHAB  B^FORK_PROC             ; Else, set address of fork process
                        00000000'GF   17  1652  4748          JMP     G^EXE$FORK              ; Schedule the fork and return
                                1658  4749
                           04 53    0E  F1  1658  4750 10$:     BBC     #XQ_CSR_V_ERR,R3,20$ ; Br if not error
;RNG0001                    10 A4    53  D0  165C     .1          MOVL    R3,CDB_L_LASTCSR(R4)    ; Save last CSR value
-1                                  05  1660  4752 20$:     RSB                             ; Return to caller
```

```
                                1661  4754          .SBTTL  FORK_PROC - Error and completion fork process handling
                                1661  4755 ;++
                                1661  4756 ; FORK_PROC - Error and completion fork processing
                                1661  4757 ;
                                1661  4758 ; Functional description:
                                1661  4759 ;
                                1661  4760 ; This routine is called as a fork process to handle errors and all completions
                                1661  4761 ; pending.
                                1661  4762 ;
                                1661  4763 ; Inputs:
                                1661  4764 ;
                                1661  4765 ;       R3 = Last CSR value
                                1661  4766 ;       R4 = CDB address
                                1661  4767 ;       R5 = CDB address
                                1661  4768 ;
                                1661  4769 ;       IPL = FIPL
                                1661  4770 ;
                                1661  4771 ; Outputs:
                                1661  4772 ;
                                1661  4773 ;       R0-R5 are destroyed.
                                1661  4774 ;
                                1661  4775 ;--
                         0819'  1661  4776          .WORD   TIMEOUT-.                       ; Offset to timeout routine
                                1663  4777 FORK_PROC::                                     ; Error/completion fork process
                                1663  4778          CLRBIT  #CDB_STS_V_FORK_PEND,CDB_B_STS(R4) ; Clear fork process flag
      20 53    0E    E1        1669  4779          BBC     #XQ_CSR_V_ERR,R3,10$            ; Br if not an error
                                166D  4780          SETBIT  #XM$V_ERR_FATAL,CDB_L_DEVDEPEND(R4) ; Indicate fatal error
  010F C4    53    90          1673  4781          MOVB    R3,CDB_L_DEVDEPEND+3(R4) ; Save low byte of CSR
55     0110 C4    D0          1678  4782          MOVL    CDB_L_UCB0(R4),R5               ; Get UCB #0 address
       0082 C5    B6          167D  4783          INCW    UCB$W_ERRCNT(R5)                ; Bump error counter
            0889  31          1681  4784          BRW     SHUTDOWN_QNA                    ; Shutdown the QNA device
                              1684  4785
                              1684  4786 3$:      ;
                              1684  4787          ; Process receive errors
                              1684  4788          ;
                         0289 30    1684  4789          BSBW    RCV_ERROR               ; Process receive error
                         007D 31    1687  4790          BRW     25$                     ; Abort messages
                              168A  4791
;RNG0001                             168A    .1 7$:
;RNG0001                             168A    .7
;RNG0001                  014B 31    168A    .8 8$:      BRW     60$                     ; Complete transmits
-1                                    168D  4793 ;
                                      168D  4794 ; Complete any TRANSMITS or RECEIVES
                                      168D  4795 ;
                                      168D  4796 10$:     PUSHQ   R6                      ; Save R6, R7
                                      1690  4797
                         050E 30      1690  4798 15$:     BSBW    ASSEM_PKTS              ; Assemble receive packets
                      F4 50   E9      1693  4799          BLBC    R0,7$                   ; Br on error or non-
                                      1696  4800          INCC    CDB_L_DBRCTR(R4)        ; Count blocks received
                50    1A A2 3C        16A0  4801          MOVZWL  CXB$W_BCNT(R2),R0       ; Get byte count
                                      16A4  4802          CNTR    R0,CDB_L_BRCCTR(R4),L   ; Count bytes received
                   16 38 A2 E9        16B0  4803          BLBC    CXB$G_R_DEST(R2),17$    ; Br if not multicast
                                      16B4  4804          INCC    CDB_L_MBLCTR(R4)        ; Count multicast blocks received
                                      16BE  4805          CNTR    R0,CDB_L_MBYCTR(R4),L   ; Count multicast bytes received
                         0E E0        16CA  4806 17$:     BBS     #RCV_STS_V_ERR,-        ; Br if FATAL receive error
                      B5 14 A2        16CC  4807                    CXB$W_R_STS(R2),3$ ;
                         0D E1        16CF  4808          BBC     #RCV_STS_V_ESETUP,-     ; Br if NOT an ESETUP receive
```

```
                 06 14 A2        16D1  4809                    CXB$W_R_STS(R2),20$     ;
                       05   E4   16D4  4810            BBSC    #CDB_STS_V_SETUP,-      ; Br if SETUP in progress and clear it
                 2D 0246 C4        16D6  4811                    CDB_B_STS(R4),25$       ;
                 51    44 A2   3C  16DA  4812 20$:     MOVZWL  CXB$W_R_PTYPE(R2),R1    ; Pick up protocol type from buffer
                                   16DE  4813
                                   16DE  4821
           0260 8F     51   B1    16DE  4822 21$:     CMPW    R1,#NI_CTR_PROTYP       ; Is this the Remote Console protocol?
                          0B   12  16E3  4823             BNEQ    22$                     ; Br if not
                          09   91  16E5  4824             CMPB    #NI_CTR_READ,-          ; Is this a read counters request?
                             46 A2        16E7  4825                    CXB$T_R_USERDAT(R2)     ;
                          05   12  16E9  4826             BNEQ    22$                     ; Br if not
                        0613   30  16EB  4827             BSBW    MOP_CTR_REQUEST         ; Else, process the request for counters
                          A0   11  16EE  4828             BRB     15$
                        13A8   30  16F0  4829 22$:      BSBW    MATCH_PROTYP            ; Try to match protocol type
                      1F 50    E8  16F3  4830             BLBS    R0,27$                  ; Br if success
           55    0210 C4     D0  16F6  4831 23$:      MOVL    CDB_L_PRMUSER(R4),R5    ; Try to get the PROMISCUOUS user
                          60   12  16FB  4832             BNEQ    34$                     ; Br if one found
                                   16FD  4833             INCC    CDB_W_UFDCTR(R4),W      ; Else, no such protocol type
                       10 A2   DD  1707  4834 25$:      PUSHL   CXB$L_LINK(R2)          ; Save next in chain
                        FD28   30  170A  4835             BSBW    ADDRCVLIST              ; Add buffer to receive list
                          52 8ED0  170D  4836             POPL    R2                      ; Restore next buffer
                          F5   12  1710  4837             BNEQ    25$                     ; Loop if more
                        FF7B   31  1712  4838             BRW     15$                     ; Look for next completed buffer
                                   1715  4839 ;
                                   1715  4840 ; If there is a promiscuous user, then copy the packet for the promiscuous
                                   1715  4841 ; user. There is a chance that the data received is not for the protocol
                                   1715  4842 ; type user just found, we will have to re-verify that the destination
                                   1715  4843 ; address is for our physical address. This is because if we are running
                                   1715  4844 ; promiscuous mode, then we will receive all packets, including those not
                                   1715  4845 ; intended for the protocol user.
                                   1715  4846 ;
                                   1715  4847 27$:      ASSUME  NMA$C_STATE_ON EQ 0
                                   1715  4848             ASSUME  NMA$C_STATE_OFF EQ 1
                 23 0247 C4     E8  1715  4849             BLBS    CDB_B_PRM(R4),32$       ; Br if hardware is NOT in promiscuous mode
                   10 38 A2     E8  171A  4850             BLBS    CXB$G_R_DEST(R2),30$    ; Br if multicast address, this
                                   171E  4851                                             ;  will be checked later.
                       38 A2    D1  171E  4852             CMPL    CXB$G_R_DEST(R2),-      ; Is this packet for this protocol
                     0256 C4        1721  4853                    CDB_G_PHYADR(R4)        ;  user?
                          D0   12  1724  4854             BNEQ    23$                     ; Br if not, don't copy packet
                       3C A2    B1  1726  4855             CMPW    CXB$G_R_DEST+4(R2),-    ; Are we sure?
                     025A C4        1729  4856                    CDB_G_PHYADR+4(R4)      ;
                          C8   12  172C  4857             BNEQ    23$                     ; Br if no, dont't copy packet
                                   172E  4858 ;
                                   172E  4859 ;       Copy the packet for the promiscuous user
                                   172E  4860 ;
                          55   DD  172E  4861 30$:      PUSHL   R5                      ; Save user's UCB address
                55    0210 C4     D0  1730  4862             MOVL    CDB_L_PRMUSER(R4),R5    ; Get PROMISCUOUS user's UCB address
                          03   13  1735  4863             BEQL    31$                     ; Br if none
                        0285   30  1737  4864             BSBW    COPY_RCV                ; Give buffer to promiscous user
                          55 8ED0  173A  4865 31$:      POPL    R5                      ; Restore user's UCB address
                                   173D  4866 ;
                                   173D  4867 ; If multicast address is the destination, then make sure that multicast
                                   173D  4868 ; address is in multicast address list for this unit.
                                   173D  4869 ;
                 1C 38 A2     E9  173D  4870 32$:      BLBC    CXB$G_R_DEST(R2),34$    ; Br if physical address
                       12CB   30  1741  4871             BSBW    MATCH_MULTI             ; Try to match multicast address
                      16 50   E8  1744  4872             BLBS    R0,34$                  ; Br if success
```

```
                        1747  4873 33$:    INCC    UCB$W_XQ_MNECTR(R5),W   ; Else, multicast not enabled
                        1751  4874         INCC    CDB_W_UFDCTR(R4),W      ; Also added in unrecognized frame dest
                AA   11 175B  4875         BRB     25$                     ; Release buffer
                        175D  4876 ;
                        175D  4877 ; If the user did not request data chaining, then check to make sure he gets
                        175D  4878 ; no chained buffers... may be user of Internal IRPs!
                        175D  4879 ;
              10 A2   D5 175D  4880 34$:    TSTL    CXB$L_LINK(R2)          ; Is this a chained message?
                 05   13 1760  4881         BEQL    36$                     ; Br if not
                        1762  4882         ASSUME  NMA$C_STATE_ON EQ 0
                        1762  4883         ASSUME  NMA$C_STATE_OFF EQ 1
         4A 00E0 C5   E8 1762  4884         BLBS    UCB$B_XQ_DCA(R5),45$    ; Br if chaining not allowed
                        1767  4885 ;
                        1767  4886 ; If there is a pending receive I/O request, complete it.
                        1767  4887 ; Otherwise, queue the buffer and, if enabled, deliver attention AST.
                        1767  4888 ;
   50    42 A5   3C 1767  4889 36$:    MOVZWL  UCB$W_DEVBUFSIZ(R5),R0  ; Get size of user's max buffer
                        176B  4890         ;&&
                        176B  4891         ;&& the following code could cause problems for an altstart user,
                        176B  4892         ;&& if the altstart user happens to receive a buffer which is 1 or 2
                        176B  4893         ;&& bytes longer than they are capable of handling. Only if the
                        176B  4894         ;&& protocol is not "padded", because the size check allows for
                        176B  4895         ;&& 2 bytes of count to be subtracted from the message size.
                        176B  4896         ;&&
                        176B  4897 ;
                        176B  4898 ; Check the size of the received buffer against what the user protocol can
                        176B  4899 ; handle.
                        176B  4900 ;
              02   A3   176B  4901         SUBW3   #XQ_C_CNTSIZ,-          ; Get the size of the receive buffer
   51    1A A2           176D  4902                 CXB$W_BCNT(R2),R1       ;  minus the count word
      50    51   B1 1770  4903         CMPW    R1,R0                   ; Is the received size larger than
                        1773  4904                                         ;  what the user can handle?
              3C   1A 1773  4905         BGTRU   45$                     ; Br if yes, error
         0192 C5   D5 1775  4906         TSTL    UCB$L_XQ_FFI(R5)        ; FAST Interface supported?
              06   13 1779  4907         BEQL    37$                     ; Br if not, standard interface
                 02D1 30 177B  4908         BSBW    FINISH_RCV_FFI          ; Else, complete FAST receive
                 FF0F 31 177E  4909         BRW     15$                     ; Look for more completions
                        1781  4910
   51    00A8 C5   9E 1781  4911 37$:    MOVAB   UCB$Q_XQ_RCVREQ(R5),R1  ; Assume that we are running in
                        1786  4912                                         ;  exclusive mode
              03   E1 1786  4913         BBC     #UCB$V_XQ_SHARE,-       ; Br if UCB is NOT in SHARED mode
      09 68 A5         1788  4914                 UCB$W_DEVSTS(R5),38$    ;
                        178B  4915 ;
                        178B  4916 ; If running in SHARED mode, then we must use the listheads in the SHR
                        178B  4917 ; data structure. We will use the source address from the received message
                        178B  4918 ; to match against the SHR structure destination address.
                        178B  4919 ;
                 01FD 30 178B  4920         BSBW    MATCH_SRC               ; Check for a match on source address
                    B7   12 178E  4921         BNEQ    33$                     ; Br if no shared user found
                        1790  4922 ;
                        1790  4923 ;       SHARED user found, use listheads in SHR data structure.
                        1790  4924 ;
   51    20 A1   9E 1790  4925         MOVAB   SHR_Q_RCVREQ(R1),R1     ; Get address of waiting IRPs
   53    00 B1   0F 1794  4926 38$:    REMQUE  @(R1),R3                ; Remove waiting IRP
              0B   1D 1798  4927         BVS     40$                     ; Br if none - queue for later
      00CC C5   50   A2 179A  4928         SUBW    R0,UCB$W_XQ_QUOTA(R5)   ; Else, lessen quota so it can be
                        179F  4929                                         ;..increased on completion
```

```
                02EB   30  179F  4930          BSBW    FINISH_RCV_IO           ; And finish the I/O
                FEEB   31  17A2  4931          BRW     15$                     ; Look for next completion
                       17A5  4932 ;
                       17A5  4933 ; Check buffer quota and queue if quota okay.
                       17A5  4934 ;
            00CC C5     50  A2  17A5  4935 40$:    SUBW    R0,UCB$W_XQ_QUOTA(R5)   ; Decrement the quota
                        1C  1E  17AA  4936         BGEQU   50$                     ; Br if we can buffer request
            00CC C5     50  A0  17AC  4937         ADDW    R0,UCB$W_XQ_QUOTA(R5)   ; Replace quota
                       17B1  4938 45$:    INCC    UCB$W_XQ_UBOCTR(R5),W   ; Else, no buffer available
                       17BB  4939         INCC    CDB_W_UBOCTR(R4),W      ; ...don't forget CDB counter
                FF3F   31  17C5  4940         BRW     25$                     ; Return buffer
                       17C8  4941
                       17C8  4942 50$:    ASSUME  UCB$Q_XQ_RCVREQ EQ UCB$Q_XQ_RCVMSG+8
                       17C8  4943         ASSUME  SHR_Q_RCVREQ EQ SHR_Q_RCVMSG+8
                51  04  C2  17C8  4944         SUBL    #4,R1                   ; Backup to backward link pointer
                       17CB  4945                                             ;  of the message queue
              00 B1  62  0E  17CB  4946         INSQUE  (R2),@(R1)              ; Queue received msg for later
                FC61   30  17CF  4947         BSBW    FILLRCVLIST             ; Try to fill the receive list
                1291   30  17D2  4948         BSBW    POKE_USER               ; Deliver ASTs
                FEB8   31  17D5  4949         BRW     15$                     ; Look for more completions
                       17D8  4950 ;
                       17D8  4951 ; NOW - scan the xmit ring entries
                       17D8  4952 ;
              0107 C4  95  17D8  4953 60$:    TSTB    CDB_B_XMTCNT(R4)        ; Any xmits in progress?
                   06  12  17DC  4954         BNEQ    70$                     ; Br if yes - look for any completed
;RNG0001        012B   31  17DE    .1         BRW     190$                    ; Else, all done
;RNG0001                17E1    .2
;RNG0001                17E1    .3 65$:
;RNG0001        0128   31  17E1    .7         BRW     190$                    ; All done
-1                     17E4  4956
          56    0105 C4  9A  17E4  4957 70$:    MOVZBL  CDB_B_LASTXMT(R4),R6    ; Get last ring entry completed
        56   009C C446  D0  17E9  4958         MOVL    CDB_L_XRINGVA(R4)[R6],R6; Get address of last ring entry
                    0F  E1  17EF  4959         BBC     #XMT_STS_V_LAST,-       ; Br if done
                 05 08 A6       17F1  4960                  XMT_W_STS(R6),75$  ;
                    0E  E0  17F4  4961         BBS     #XMT_STS_V_ERR,-        ; Br if not done
                 E8 08 A6       17F6  4962                  XMT_W_STS(R6),65$  ;  ..leave
                    0A A6  B5  17F9  4963 75$:    TSTW    XMT_Q_TDR(R6)           ; Are we really done?
                    E3  13  17FC  4964         BEQL    65$                     ; Br if not!
              53    00E4 D4  0F  17FE  4965         REMQUE  @CDB_Q_XMTPND(R4),R3    ; Get next XMIT CXB
                    DC  1D  1803  4966         BVS     65$                     ; Br if none there (yet)
                    04  12  1805  4967         BNEQ    77$                     ; Br if more entries on queue
                 020A C4  94  1807  4968         CLRB    CDB_B_TIM_XMT(R4)       ; Stop the xmit timer
                 0107 C4  97  180B  4969 77$:    DECB    CDB_B_XMTCNT(R4)        ; One less transmit pending
                 0105 C4  96  180F  4970         INCB    CDB_B_LASTXMT(R4)       ; Bump ring pointer
                   FC 8F  8A  1813  4971         BICB    #^C<MAX_C_XMT-1>,-      ; Modulo receive ring entry size
                 0105 C4        1816  4972                 CDB_B_LASTXMT(R4)       ;       ...
                       1819  4973 ;
                       1819  4974 ; Transmit complete
                       1819  4975 ;
                       1819  4976         CLRBIT  XMT_DSC_V_VALID,-       ; Indicate that buffer is not valid
                       1819  4977                 XMT_W_ADDRHI(R6)        ;
                    01 A6  90  181E  4978         MOVB    XMT_Q_FLAG+1(R6),-      ; Save diagnostic return info
                 0119 C4        1821  4979                 CDB_B_DIAG1(R4)         ;
                    08 A6  B0  1824  4980         MOVW    XMT_Q_STS(R6),-         ; Save diagnostic return info
                 011A C4        1827  4981                 CDB_Q_DIAG2(R4)         ;
              55    24 A3  D0  182A  4982         MOVL    CXB$L_T_UCB(R3),R5      ; Get (presumed) UCB address
                       182E  4983         ASSUME  CXB$L_T_UCB  EQ  CXB$L_T_IRP
```

```
                04 55    00    E4  182E  4984          BBSC    #0,R5,80$                       ; Fix up address & BR if UCB address
                55   1C A5    D0  1832  4985          MOVL    IRP$L_UCB(R5),R5                ; Else, get real UCB address
                                 1836  4986
                52   22 A3    9A  1836  4987 80$:     MOVZBL  CXB$B_XQ_SLOT(R3),R2            ; Get mapping slot number used
                                 183A  4988          CLRBIT  R2,CDB_B_XMTMAP(R4)             ; Clear in use flag
                                 183F  4989
                                 183F  4990          CPUDISP <<790,90$>,-
                                 183F  4991                   <780,90$>,-
                                 183F  4992                   <750,90$>,-
                                 183F  4993                   <730,90$>,-
;RNG0001                         183F    .1                   <UV2,90$>,-
;RNG0001                         183F    .2                   <UV1,100$>>                     ; Skip map registers if MicroVAX I
;RNG0001                         185B    .3
;RNG0001                         185B    .4 90$:     ;  All but MicroVAX I
;RNG0001                         185B    .5
;RNG0001         51   24 A5    D0  185B    .6          MOVL    UCB$L_CRB(R5),R1                ; Get CRB address
-3                               185F  4997          ASSUME  VEC$B_NUMREG EQ VEC$W_MAPREG+2
                                 185F  4998          ASSUME  VEC$B_DATAPATH EQ VEC$B_NUMREG+1
                    38 A442      D0  185F  4999          MOVL    CDB_L_XMTMAP(R4)[R2],-  ; Setup map register data
                       34 A1         1863
                                 1865  5000                   CRB$L_INTD+VEC$W_MAPREG(R1) ;  and data path number
                          52     95  1865  5001          TSTB    R2                              ; Was it the pre-allocated one?
                          0D     13  1867  5002          BEQL    95$                             ; Br if yes - clear data path number
                38 A442   01     CE  1869  5003          MNEGL   #1,CDB_L_XMTMAP(R4)[R2]       ; Indicate map register not allocated
                                 186E  5004          RELMPR                                  ; Release the map registers
                          03     11  1874  5005          BRB     100$                            ; Complete the request
                                 1876  5006
                                 1876  5007 95$:     ASSUME  VEC$B_DATAPATH EQ VEC$W_MAPREG+3
                       3B A4     94  1876  5008          CLRB    CDB_L_XMTMAP+3(R4)              ; Clear data path number used
                50   1A A3     3C  1879  5009 100$:    MOVZWL  CXB$W_BCNT(R3),R0               ; Get byte count of message
                   51   50     D0  187D  5010          MOVL    R0,R1                           ; Copy length for accounting
                                 1880  5011          ;
                                 1880  5012          ; Perform accounting for the QNA
                                 1880  5013          ;
                                 1880  5014          INCC    CDB_L_DBSCTR(R4)                ; Count blocks sent
                                 188A  5015          CNTR    R1,CDB_L_BSNCTR(R4),L           ; Count bytes sent
             50     50   10     78  1896  5016          ASHL    #16,R0,R0                       ; Move to high word
                    50   01     B0  189A  5017          MOVW    S^#SS$_NORMAL,R0                ; Set completion status
                         0C     E4  189D  5018          BBSC    #XMT_DSC_V_SETUP,-              ; Br if SETUP operation
                      42 02 A6      189F  5019                  XMT_Q_ADDR[I](R6),120$          ;
          05 08 A6      0E     E1  18A2  5020          BBC     #XMT_STS_V_ERR,XMT_W_STS(R6),110$ ; Br if not a FATAL error
                           0092     30  18A7  5021          BSBW    XMT_ERROR                       ; Process XMIT error
                             38     11  18AA  5022          BRB     120$                            ; Skip the accounting
                                 18AC  5023          ;
                                 18AC  5024          ; Perform accounting on a per protocol type basis and on unit.
                                 18AC  5025          ;
                                 18AC  5026 110$:    CNTR    R1,UCB$L_XQ_SBYCTR(R5),L        ; Bump the bytes sent counter
                                 18B8  5027          INCC    UCB$L_XQ_SBLCTR(R5),L           ; Bump the blocks sent counter
                          04     EF  18C2  5028          EXTZV   #XMT_STS_V_COL,-                ; Get number of collisions
                08 A6     04         18C4  5029                  #XMT_STS_S_COL,XMT_W_STS(R6),R1
                          51         18C7
                          1A     13  18C8  5030          BEQL    120$                            ; Br if none
                          51     97  18CA  5031          DECB    R1                              ; More than one?
                          0C     13  18CC  5032          BEQL    115$                            ; Br if only one
                                 18CE  5033          INCC    CDB_L_BSMCTR(R4)                ; Count blocks sent with multiple errors
                          0A     11  18D8  5034          BRB     120$                            ; Continue
                                 18DA  5035 115$:    INCC    CDB_L_BS1CTR(R4)                ; Count blocks sent with 1 error
```

```
                          18E4  5036
      1B 24 A3    E8      18E4  5037 120$:   BLBS    CXB$L_T_IRP(R3),130$    ; BR if FAST interface CXB
    53    24 A3    D0      18E8  5038         MOVL    CXB$L_T_IRP(R3),R3      ; Else, get IRP address
             50    DD      18EC  5039         PUSHL   R0                      ; Save status return
 50     0094 C3    D0      18EE  5040         MOVL    IRP$L_XQ_SETUP(R3),R0   ; Is there a SETUP mode buffer
             06    13      18F3  5041         BEQL    125$                    ; Br if none
 00000000'GF       16      18F5  5042         JSB     G^COM$DRVDEALMEM        ; Deallocate the buffer
             50 8ED0       18FB  5043 125$:   POPL    R0                      ; Restore status return
           024F    30      18FE  5044         BSBW    IO_DONE                 ; Finish the I/O request
             03    11      1901  5045         BRB     170$                    ; Continue in common code
           013B    30      1903  5046 130$:   BSBW    FINISH_XMT_FFI          ; Else, complete FFI XMIT
           EB7E    30      1906  5047 170$:   BSBW    XMT_ALT_START           ; Start up any other transmits
           FECC    31      1909  5048         BRW     60$                     ; Look for next completion
                           190C  5049
                           190C  5050 190$:   POPQ    R6                      ; Restore R6, R7
                       05  190F  5051         RSB
                           1910  5052
```

```
                     1910  5054          .SBTTL  RCV_ERROR - Process receive errors
                     1910  5055          .SBTTL  XMT_ERROR - Process transmit errors
                     1910  5056 ;++
                     1910  5057 ; RCV_ERROR - Process receive errors
                     1910  5058 ;
                     1910  5059 ; Functional description:
                     1910  5060 ;
                     1910  5061 ; This routine adjusts all appropriate counters and checks all errors.
                     1910  5062 ;
                     1910  5063 ; Inputs:
                     1910  5064 ;       R2 = CXB address
                     1910  5065 ;       R4 = CDB address
                     1910  5066 ;
                     1910  5067 ; Outputs:
                     1910  5068 ;       none.
                     1910  5069 ;
                     1910  5070 ;--
                     1910  5071
                     1910  5072 RCV_ERROR:
                     1910  5073          INCC    CDB_W_RFLCTR(R4),W      ; Count receive failures
          01   E1    191A  5074          BBC     #RCV_STS_V_CRCERR,-     ; Br if not a CRC error
06 14 A2             191C  5075                  CXB$Q_R_STS(R2),20$     ;
                     191F  5076          SETBIT  #0,CDB_Q_RFLMAP(R4)     ; Indicate CRC error
          02   E1    1925  5077 20$:     BBC     #RCV_STS_V_FRAME,-      ; Br if not a framing error
C6 14 A2             1927  5078                  CXB$Q_R_STS(R2),40$     ;
                     192A  5079          SETBIT  #1,CDB_Q_RFLMAP(R4)     ; Indicate FRAME error
          0B   E1    1930  5080 40$:     BBC     #RCV_STS_V_RUNT,-       ; Br if not a RUNT packet
06 14 A2             1932  5081                  CXB$Q_R_STS(R2),90$     ;
                     1935  5082          SETBIT  #2,CDB_Q_RFLMAP(R4)     ; Indicate FRAME error
               05    193B  5083 90$:     RSB
                     193C  5084
                     193C  5085
                     193C  5086 ;++
                     193C  5087 ; XMT_ERROR - Process transmit errors
                     193C  5088 ;
                     193C  5089 ; Functional description:
                     193C  5090 ;
                     193C  5091 ; This routine adjusts all appropriate counters and checks all errors.
                     193C  5092 ;
                     193C  5093 ; Inputs:
                     193C  5094 ;       R4 = CDB address
                     193C  5095 ;       R6 = Transmit ring entry address
                     193C  5096 ;
                     193C  5097 ; Outputs:
                     193C  5098 ;       R0 = error code
                     193C  5099 ;
                     193C  5100 ;--
                     193C  5101
                     193C  5102 XMT_ERROR:
                     193C  5103          INCC    CDB_W_SFLCTR(R4),W      ; Count send failures
50   20C4 8F    B0   1946  5104          MOVW    #SS$_COMMHARD,R0        ; Assume No Carrier failure
          0F     E1   194B  5105          BBC     #XMT_STS_V_ABORT,-      ; Br if NOT 16 retries failed
     0B 08 A6          194D  5106                  XMT_Q_STS(R6),20$       ;
50    0334 8F    B0   1950  5107          MOVW    #SS$_DEVREQERR,R0       ; Else, DEVREQERR error
                     1955  5108          SETBIT  #0,CDB_W_SFLMAP(R4)     ; Set bitmap
          0C     E1   195B  5109 20$:     BBC     #XMT_STS_V_LCAR,-       ; Br if NOT Loss of Carrier
     0B 08 A6          195D  5110                  XMT_Q_STS(R6),40$       ;
```

```
                50   204C 8F    B0  1960  5111          MOVW    #SS$_DISCONNECT,R0      ; Else, DISCONNECT error
                                    1965  5112          SETBIT  #1,CDB_W_SFLMAP(R4)     ; Set bitmap
                          08    E1  196B  5113 40$:      BBC     #XMT_STS_V_FAIL,-       ; Br if NOT Collision check failure
                     0F 08 A6         196D  5114                  XMT_Q_STS(R6),60$
                50   005C 8F    B0  1970  5115          MOVW    #SS$_DATACHECK,R0       ; Else, DATACHECK error
                                    1975  5116          INCC    CDB_Q_CDCCTR(R4),W      ; Count separate counter
                          0B    E1  197F  5117 60$:      BBC     #XMT_STS_V_NOCAR,-      ; Br if NOT Carrier failure
                     06 08 A6         1981  5118                  XMT_Q_STS(R6),90$       ;
                                    1984  5119          SETBIT  #2,CDB_W_SFLMAP(R4)     ; Se  bitmap
                                05  198A  5120 90$:      RSB
```

```
                                1980  5122          .SBTTL   SUBROUTINES TO FIND SHR MATCH ON SOURCE ADDRESS
                                1980  5123 ;++
                                1980  5124 ; Functional description:
                                1980  5125 ;
                                1980  5126 ; Subroutine to find SHR data structure for user
                                1980  5127 ;
                                1980  5128 ; Inputs:
                                1980  5129 ;       R2 = Receive CXB address
                                1980  5130 ;       R5 = UCB address
                                1980  5131 ;
                                1980  5132 ; Outputs:
                                1980  5133 ;       R1 = Address if SHR data structure if match
                                1980  5134 ;       All other registers preserved.
                                1980  5135 ;       Z-Bit set then match.
                                1980  5136 ;       Z-Bit clear then no match.
                                1980  5137 ;--
                                1980  5138
                                1980  5139 MATCH_SRC:                                    ; Try to find shared user
                    05    BB    1980  5140          PUSHR   #^M<R0,R2>                  ; Save registers
                                198D  5141 ;
                                198D  5142 ; Try to find match among limited shared users of protocol type
                                198D  5143 ;
          50  0098  C5    9E    198D  5144          MOVAB   UCB$Q_XQ_SHARE(R5),R0       ; Save address of listhead
              51    50    D0    1992  5145          MOVL    R0,R1                       ; Copy listhead address
                                1995  5146          ASSUME  SHR_L_QFL EQ 0
              51    61    D0    1995  5147 20$:     MOVL    (R1),R1                     ; Get next in list
              50    51    D1    1998  5148          CMPL    R1,R0                       ; Back to start of list?
                    06    13    199B  5149          BEQL    30$                         ; Br if yes - no source match
                    13    10    199D  5150          BSBB    CHECK_SRC                   ; Check for match
                    F4    12    199F  5151          BNEQ    20$                         ; Br if none
                    0C    11    19A1  5152          BRB     50$                         ; Return in success (Z-bit is set)
                                19A3  5153 ;
                                19A3  5154 ; No match on limited users - try to use default user
                                19A3  5155 ;
          51  00C4  C5    D0    19A3  5156 30$:     MOVL    UCB$L_XQ_DEFUSR(R5),R1      ; Get address of default user
                    02    13    19A8  5157          BEQL    40$                         ; Br if no default user
                    50    D4    19AA  5158          CLRL    R0                          ; Return success
              50    50    D0    19AC  5159 40$:     MOVL    R0,R0                       ; Return success/failure indicator
                    05    BA    19AF  5160 50$:     POPR    #^M<R0,R2>                  ; Restore registers, don't reset Z-BIT
                    05    19B1  5161          RSB
                                19B2  5162
                                19B2  5163 ;++
                                19B2  5164 ; Functiona  description:
                                19B2  5165 ;
                                19B2  5166 ; Subroutine to check if source address in message matches SHR address
                                19B2  5167 ;
                                19B2  5168 ; Inputs:
                                19B2  5169 ;       R1 = Address of SHR
                                19B2  5170 ;       R2 = Address of MSG buffer
                                19B2  5171 ;
                                19B2  5172 ; Outputs:
                                19B2  5173 ;       Z-Bit set then match.
                                19B2  5174 ;       Z-Bit clear then no match.
                                19B2  5175 ;--
                                19B2  5176
                                19B2  5177 CHECK_SRC:                                   ; Check for match with SHR data base
      12  A1    3E  A2    D1    19B2  5178          CMPL    CXB$G_R_SRC(R2),SHR_G_DEST(R1) ; Source address match?
```

```
                05   12  19B7  5179          BNEQ    10$                         ; Br if no - try for next
            42  A2   B1  19B9  5180          CMPW    CXB$G_R_SRC+4(R2),-         ; Really match?
            16  A1       19BC  5181                   SHR_G_DEST+4(R1)            ;       ...
                05   19BE  5182 10$:        RSB                                 ; Return to caller
```

```
                                19BF  5184          .SBTTL  COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
                                19BF  5185 ;++
                                19BF  5186 ; COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
                                19BF  5187 ;
                                19BF  5188 ; Functional description:
                                19BF  5189 ;
                                19BF  5190 ; This routine allocates a receive buffer in which to copy the a receive
                                19BF  5191 ; buffer for the PROMISCUOUS user.
                                19BF  5192 ;
                                19BF  5193 ;  Inputs:
                                19BF  5194 ;
                                19BF  5195 ;       R2 = Receive CXB address
                                19BF  5196 ;       R4 = CDB address
                                19BF  5197 ;       R5 = UCB address of PROMISCUOUS user
                                19BF  5198 ;       R6 = Address of receive ring entry
                                19BF  5199 ;
                                19BF  5200 ;       IPL = FIPL
                                19BF  5201 ;
                                19BF  5202 ;  Outputs:
                                19BF  5203 ;
                                19BF  5204 ;       R0,R1,R3 are destroyed.
                                19BF  5205 ;       All other registers are preserved.
                                19BF  5206 ;--
                                19BF  5207 COPY_RCV:                                    ; Copy the xmit buffer to rcv buffer
                    52    DD    19BF  5208          PUSHL   R2                        ; Save address of original buffer
                1C A2     01    B1    19C1  5209          CMPW    #1,CXB$W_R_NCHAIN(R2)     ; Is there more than 1 in chain?
                          76    12    19C5  5210          BNEQ    80$                       ; Br if yes - we can only handle 1
                51    1A  A2    3C    19C7  5211          MOVZWL  CXB$W_BCNT(R2),R1         ; Get size of buffer
                00000048  8F    C0    19CB  5212          ADDL    #CXB$C_HEADER,R1          ; Compute size of needed buffer
                          51          19D1
                00000000'GF   16    19D2  5213          JSB     G^EXE$ALONONPAGED         ; Allocate a receive buffer
                      62  50    E9    19D8  5214          BLBC    R0,80$                    ; Br if failure
                    53    6E    D0    19DB  5215          MOVL    (SP),R3                   ; Get address of original buffer
                          36    BB    19DE  5216          PUSHR   #^M<R1,R2,R4,R5>          ; Save registers
                62    63    51    28    19E0  5217          MOVC3   R1,(R3),(R2)              ; Copy everything to new buffer
                          36    BA    19E4  5218          POPR    #^M<R1,R2,R4,R5>          ; Save registers
                  08  A2    51    B0    19E6  5219          MOVW    R1,CXB$W_SIZE(R2)         ; Reset size field
                                19EA  5220 ;
                                19EA  5221 ; If there is a pending receive I/O request, complete it.
                                19EA  5222 ; Otherwise, queue the buffer and, if enabled, deliver attention AST.
                                19EA  5223 ;
                  51    42  A5    3C    19EA  5224          MOVZWL  UCB$W_DEVBUFSIZ(R5),R1    ; Get the user's buffer size
                53    00A8  D5    0F    19EE  5225          REMQUE  @UCB$Q_XQ_RCVREQ(R5),R3   ; Remove waiting IRP
                          0A    1D    19F3  5226          BVS     40$                       ; Br if none - queue for later
                00CC  C5    51    A2    19F5  5227          SUBW    R1,UCB$W_XQ_QUOTA(R5)     ; Else, lessen quota so it can be
                                19FA  5228                                                ;..increased on completion
                        008D  30    19FA  5229          BSBW    FINISH_RCV_IO             ; And finish the I/O
                          3E    11    19FD  5230          BRB     80$                       ; Look for next completion
                                19FF  5231 ;
;RNG0001                         19FF    .1 ; Check buffer quota and queue if quota okay.
;RNG0001                         19FF    .2 ;
;RNG0001         00CC  C5    51    A2    19FF    .3 40$:     SUBW    R1,UCB$W_XQ_QUOTA(R5)     ; Decrement the quota
;RNG0001                   2C    1E    1A04    .4          BGEQU   60$                       ; Br if we can buffer request
;RNG0001                         1A06    .5          ;
;RNG0001                         1A06    .6          ; Return buffer to pool if user buffer failure
;RNG0001                         1A06    .7          ;
;RNG0001         00CC  C5    51    A0    1A06    .8          ADDW    R1,UCB$W_XQ_QUOTA(R5)     ; Replace quota
```

```
;RNG0001                                1A0B    .9          INCC    UCB$W_XQ_UBUCTR(R5),W   ; Else, no buffer available
;RNG0001                                1A15    .10         INCC    CDB_W_UBOCTR(R4),W      ; ...don't forget CDB counter
;RNG0001                          DD    1A1F    .11 50$:    PUSHL   CXB$L_LINK(R2)          ; Save next in chain
;RNG0001                       52 D0    1A22    .12         MOVL    R2,R0                   ; Copy the buffer address
;RNG0001                00000000'GF 16    1A25    .13         JSB     G^COM$DRVDEALMEM        ; DEALLOCATE the buffer
;RNG0001                    52 8ED0      1A2B    .14         POPL    R2                      ; Restore next buffer
;RNG0001                    EF 12        1A2E    .15         BNEQ    50$                     ; Loop if more
;RNG0001                    0B 11        1A30    .16         BRB     80$                     ; Else, leave
;RNG0001                                1A32    .17
;RNG0001                00A4 D5 62 0E    1A32    .18 60$:    INSQUE  (R2),@UCB$Q_XQ_RCVMSG+4(R5) ; Queue received msg for later
-19                          F919 30     1A37    5251        BSBW    FILLR_VLIST             ; Try to fill the receive list
                            1029 30      1A3A    5252        BSBW    POKE_USER               ; Deliver ASTs
                              52 8ED0    1A3D    5253 80$:   POPL    R2                      ; Restore R2
                                 05      1A40    5254        RSB                             ; Return to caller
```

```
                     1A41  5256          .SBTTL  FINISH_XMT_FFI - Finish FAST interface transmit processing
                     1A41  5257 ;++
                     1A41  5258 ; FINISH_XMT_FFI - Finish FAST interface transmit processing
                     1A41  5259 ;
                     1A41  5260 ; Functional description:
                     1A41  5261 ;
                     1A41  5262 ; This routine completes a transmit CXB for a particular user of the fast
                     1A41  5263 ; interface.
                     1A41  5264 ;
                     1A41  5265 ;  Inputs:
                     1A41  5266 ;
                     1A41  5267 ;       R0 = Status of transmit request
                     1A41  5268 ;       R3 = transmit CXB address
                     1A41  5269 ;       R4 = CDB address
                     1A41  5270 ;       R5 = UCB address
                     1A41  5271 ;
                     1A41  5272 ;       IPL = FIPL
                     1A41  5273 ;
                     1A41  5274 ;  Outputs:
                     1A41  5275 ;
                     1A41  5276 ;       R0-R3 are destroyed.
                     1A41  5277 ;       All other registers are preserved.
                     1A41  5278 ;
                     1A41  5279 ;--
                     1A41  5280          ASSUME  IPL$_SYNCH EQ IPL$_XQ_FIPL
                     1A41  5281 FINISH_XMT_FFI::                             ; Finish FAST interface transmit reques*
           54    DD  1A41  5282          PUSHL   R4                          ; Save R4
54   0192 C5    DO  1A43  5283          MOVL    UCB$L_XQ_FFI(R5),R4         ; Get FFI block address
     14 B4    16  1A48  5284          JSB     @FFI$L_XMIT_DONE(R4)        ; Call back the user with CXB
        54 8ED0  1A4B  5285          POPL    R4                          ; Restore R4
           05  1A4E  5286          RSB
```

```
                     1A4F  5288          .SBTTL  FINISH_RCV_FFI - Finish FAST receive processing
                     1A4F  5289 ;++
                     1A4F  5290 ; FINISH_RCV_FFI - Finish FAST receive processing
                     1A4F  5291 ;
                     1A4F  5292 ; Functional description:
                     1A4F  5293 ;
                     1A4F  5294 ; This routine completes a receive CXB for a particular user of the fast
                     1A4F  5295 ; interface.
                     1A4F  5296 ;
                     1A4F  5297 ;  Inputs:
                     1A4F  5298 ;
                     1A4F  5299 ;       R2 = receive CXB address
                     1A4F  5300 ;       R4 = CDB address
                     1A4F  5301 ;       R5 = UCB address
                     1A4F  5302 ;
                     1A4F  5303 ;       IPL = FIPL
                     1A4F  5304 ;
                     1A4F  5305 ;  Outputs:
                     1A4F  5306 ;
                     1A4F  5307 ;       R0-R3 are destroyed.
                     1A4F  5308 ;       All other registers are preserved.
                     1A4F  5309 ;
                     1A4F  5310 ;--
                     1A4F  5311          ASSUME  IPL$_SYNCH EQ IPL$_XQ_FIPL
                     1A4F  5312 FINISH_RCV_FFI::                         ; Finish FAST recieve request
     54    DD        1A4F  5313          PUSHL   R4                      ; Save R4
53   52    D0        1A51  5314          MOVL    R2,R3                   ; Copy CXB address
  3E A3    7D        1A54  5315          MOVC    CXB$G_R_SRC(R3),-       ; Copy source address
  28 A3              1A57  5316                  CXB$Q_STATION(R3)       ;                    ...
0046 8F    B0        1A59  5317          MOVW    #CXB$T_R_USERDAT,CXB$W_BOFF(R3) ; Set offset to received data
  18 A3              1A5D
                     1A5F  5318          ASSUME  NMA$C_STATE_ON EQ 0
                     1A5F  5319          ASSUME  NMA$C_STATE_OFF EQ 1
  09 00DD C5    E8   1A5F  5320          BLBS    UCB$B_XQ_PAD(R5),30$    ; Br if padding is disabled
     46 A3    B0   1A64  5321          MOVW    CXB$W_R_SIZE(R3),-      ; Else, set real size of buffer
     1A A3         1A67  5322                  CXB$W_BCNT(R3)
  18 A3    02    A0   1A69  5323          ADDW    #XQ_C_CNTSIZ,CXB$W_BOFF(R3) ; Adjust offset
50   010C C4    D0   1A6D  5324 30$:     MOVL    CDB_L_DEVDEPEND(R4),R0  ; Set controller bits
 50   44 A5    C8   1A72  5325          BISL    UCB$L_DEVDEPEND(R5),R0  ; Set status flags
54   0192 C5    D0   1A76  5326          MOVL    UCB$L_XQ_FFI(R5),R4     ; Get FFI block address
     18 B4    16   1A7B  5327          JSB     @FFI$C_RECV_DONE(R4)    ; Call back the user with CXB
        54 8ED0   1A7E  5328          POPL    R4                      ; Restore R4
     52   53    D0   1A81  5329          MOVL    R3,R2                   ; Was buffer consumed?
          03    13   1A84  5330          BEQL    90$                     ; Br if YES
        F9AC    30   1A86  5331          BSBW    ADDRCVLIST              ; Else, add buffer to receive list
                05   1A89  5332 90$:     RSB
```

```
                        1A8A  5334          .SBTTL  FINISH_RCV_IO - Finish receive I/O processing
                        1A8A  5335 ;++
                        1A8A  5336 ; FINISH_RCV_IO - Finish receive I/O processing
                        1A8A  5337 ;
                        1A8A  5338 ; Functional description:
                        1A8A  5339 ;
                        1A8A  5340 ; This routine completes a receive operation that has been matched with a
                        1A8A  5341 ; message block. After the receive has been completed the message free list
                        1A8A  5342 ; is filled and a receive is started if needed.
                        1A8A  5343 ;
                        1A8A  5344 ;  Inputs:
                        1A8A  5345 ;
                        1A8A  5346 ;       R2 = receive CXB address
                        1A8A  5347 ;       R3 = I/O packet address
                        1A8A  5348 ;       R4 = CDB address
                        1A8A  5349 ;       R5 = UCB address
                        1A8A  5350 ;
                        1A8A  5351 ;       IPL = FIPL
                        1A8A  5352 ;
                        1A8A  5353 ;  Outputs:
                        1A8A  5354 ;
                        1A8A  5355 ;       R5 is reset to UCB address from IRP
                        1A8A  5356 ;
                        1A8A  5357 ;       The request is completed via I/O post.
                        1A8A  5358 ;--
                        1A8A  5359
                        1A8A  5360 FINISH_RCV_IO::                          ; Finish recieve I/O request
2C A3    52    D0       1A8A  5361          MOVL    R2,IRP$L_SVAPTE(R3)     ; Save block address
     3E A2    7D       1A8E  5362          MOVQ    CXB$G_R_SRC(R2),-       ; Copy source address for DECnet
     40 A3             1A91  5363                  IRP$Q_STATION(R3)       ;       ...
        07    E1       1A93  5364          BBC     #IRP$V_DIAGBUF,-        ; Br if no diagnotic buffer
  27 2A A3             1A95  5365                  IRP$W_STS(R3),10$       ;
51    4C A3    D0       1A98  5366          MOVL    IRP$L_DIAGBUF(R3),R1    ; Get diagnostic buffer
50    0C A1    9E       1A9C  5367          MOVAB   RHDR_T_DATA(R1),R0      ; Assume this is just a read header
08 A1    1A    B1       1AA0  5368          CMPW    #RHDR_C_LENGTH,DIAG_W_SIZE(R1) ; Is this just a header buffer?
           10    13       1AA4  5369          BEQL    5$                      ; Br if yes
50    30 A1    9E       1AA6  5370          MOVAB   DIAG_T_RDATA(R1),R0     ; Else, must be a diagnostic buffer
     0B A2    90       1AAA  5371          MOVB    CXB$B_R_FLAGS(R2),-     ; Save diagnostic return info
   0119 C4             1AAD  5372                   CDB_B_DIAG1(R4)        ;       ...
     14 A2    B0       1AB0  5373          MOVW    CXB$W_R_STS(R2),-       ;       ...
   011A C4             1AB3  5374                   CDB_W_DIAG2(R4)        ;       ...
           3C    BB       1AB6  5375 5$:      PUSHR   #^M<R2,R3,R4,R5>        ; Save registers
38 A2      0E    28       1AB8  5376          MOVC3   #RHDR_C_DATA,CXB$T_R_DATA(R2),(R0) ; Move header info
           60             1ABC
           3C    BA       1ABD  5377          POPR    #^M<R2,R3,R4,R5>        ; Restore registers
62    46 A2    9E       1ABF  5378 10$:     MOVAB   CXB$T_R_USERDAT(R2),(R2); Set address of received data
04 A2    3C A3    D0       1AC3  5379          MOVL    IRP$L_XQ_DATBUF(R3),4(R2); Set address of user buffer
     42 A5    A0       1AC8  5380          ADDW    UCB$W_DEVBUFSIZ(R5),-   ; Adjust receive buffer quota
   00CC C5             1ACB  5381                   UCB$W_XQ_QUOTA(R5)     ;
51    1A A2    3C       1ACE  5382          MOVZWL  CXB$W_BCNT(R2),R1       ; Find length of received message
                        1AD2  5383 ;
                        1AD2  5384 ; Perform accounting on a per protocol type basis.
                        1AD2  5385 ;
                        1AD2  5386          CNTR    R1,UCB$L_XQ_RBYCTR(R5),L ; Bump the bytes received counter
                        1ADE  5387          INCC    UCB$L_XQ_RBLCTR(R5),L   ; Bump the blocks received counter
                        1AE8  5388 ;
                        1AE8  5389 ; If padding is enabled, then the size of the data is contained in the
```

```
                                1AE8  5390 ; message as the first word of data.
                                1AE8  5391 ;
                                1AE8  5392         ASSUME  NMA$C_STATE_ON EQ 0
                                1AE8  5393         ASSUME  NMA$C_STATE_OFF EQ 1
                     OF 00DD C5 E8  1AE8  5394         BLBS    UCB$B_XQ_PAD(R5),15$    ; Br if padding is disabled
                     50   46 A2 3C  1AED  5395         MOVZWL  CXB$W_R_SIZE(R2),R0     ; Else, pick up real size of
                                1AF1  5396                                             ;  MSG from the message itself
                        62   02 C0  1AF1  5397         ADDL    #XQ_C_CNTSIZ,(R2)       ; Move pointer past the count field
                                1AF4  5398 ;
                                1AF4  5399 ; Verify that the 1st word of data at least makes some sense. The byte
                                1AF4  5400 ; count for the message must be less than the size of the entire received
                                1AF4  5401 ; message.
                                1AF4  5402 ;
                        51   50 B1 1AF4  5403         CMPW    R0,R1                   ; Is size field larger than buffer?
                             31 1E 1AF7  5404         BGEQU   45$                     ; Br if yes -
                                1AF9  5405                                             ;  must be strictly Less Than, because
                                1AF9  5406                                             ;  the size field is 2 extra bytes.
                        51   50 3C 1AF9  5407         MOVZWL  R0,R1                   ; Else, copy the real buffer size
                                1AFC  5408
                  3A A3    01 9B 1AFC  5409 15$:     MOVZBW  S^#SS$_NORMAL,IRP$W_XQ_STATUS(R3) ; Assume success
                  32 A3    51 B1 1B00  5410         CMPW    R1,IRP$W_BCNT(R3)       ; Request larger than user buffer size?
                             0A 1B 1B04  5411         BLEQU   20$                     ; Br if no - okay
                  51    32 A3 3C 1B06  5412         MOVZWL  IRP$W_BCNT(R3),R1       ; Else, set size to minimum of two
                     0838 8F B0 1B0A  5413         MOVW    #SS$_DATAOVERUN,IRP$W_XQ_STATUS(R3) ; Set return status
                        3A A3       1B0E
                                1B10  5414 ;
                                1B10  5415 ; If chained buffers, then setup IRP$W_STS and don't reset the USER BUFFER
                                1B10  5416 ; SIZE (IRP$W_BCNT), because chained buffers need to have the USER BUFFER
                                1B10  5417 ; SIZE in IRP$W_BCNT.
                                1B10  5418 ;
                           10 A2 D5 1B10  5419 20$:      TSTL    CXB$L_LINK(R2)          ; Is this a complex chained buffer?
                              0B 13 1B13  5420         BEQL    30$                     ; Br if no - set transfer size
                              05 E1 1B15  5421         BBC     #IRP$V_CHAINED,-        ; Br if user cannot accept complex
                        10 2A A3    1B17  5422                  IRP$W_STS(R3),45$       ;  chained buffers
                              08 A8 1B1A  5423         BISW    #IRP$M_COMPLX,-         ; Else indicate complex
                           2A A3    1B1C  5424                  IRP$W_STS(R3)           ;    chained buffers
                              04 11 1B1E  5425         BRB     40$                     ; And don't change user buffer size
                     32 A3    51 B0 1B20  5426 30$:     MOVW    R1,IRP$W_BCNT(R3)       ; Set size of transfer
                  50    51    10 78 1B24  5427 40$:     ASHL    #16,R1,R0               ; Set buffer size in status
                              06 12 1B28  5428         BNEQ    50$                     ; Br if success
                        0054 8F B0 1B2A  5429 45$:     MOVW    #SS$_CTRLERR,-          ; Set data transfer error
                           3A A3    1B2E  5430                  IRP$W_XQ_STATUS(R3)     ;
                                1B30  5431
                     50    3A A3 B0 1B30  5432 50$:     MOVW    IRP$W_XQ_STATUS(R3),R0  ; Get status
                              1A 10 1B34  5433 60$:     BSBB    IO_DONE                 ; Post the I/O request
                           F8FA 31 1B36  5434         BRW     FILRCVLIST              ; Fill up the receive buffers
                                1B39  5435 ;
                                1B39  5436 ; Complete an I/O request packet
                                1B39  5437 ;
                                1B39  5438 ABORT_PKT:                                  ; Abort the I/O request
                           50   2C 9A 1B39  5439         MOVZBL  S^#SS$_ABORT,R0         ; Return aborted status
                                1B3C  5440 IO_DONE3:                                   ; Complete the I/O request, check for
                                1B3C  5441                                             ;    timeout
                     55    1C A3 D0 1B3C  5442         MOVL    IRP$L_UCB(R3),R5        ; Get UCB address
;RNG0001              51   00C8 C5 D0 1B40     .1      MOVL    UCB$L_XQ_CDB(R5),R1     ; Get CDB address
-2                   010C C1   09 E1 1B45  5445         BBC     #XM$V_STS_TIMO,CDB_L_DEVDEPEND(R1),IO_DONE ; Br if not a timeout
                                05 1B4A
```

```
                50    022C 8F    3C   1B4B  5446          MOVZWL  #SS$_TIMEOUT,R0          ; Else, return real error code
                                        1B50  5447 IO_DONE:                              ; Complete an I/O request
                  38 A3    50    D0   1B50  5448          MOVL    R0,IRP$L_IOST1(R3)      ; Set status return and size
                                        1B54  5449 IO_DONE1:                             ; Alternate entry point
                   55    1C A3    D0   1B54  5450          MOVL    IRP$L_UCB(R3),R5        ; Get UCB address
                3C A3    44 A5    D0   1B58  5451          MOVL    UCB$L_DEVDEPEND(R5),IRP$L_IOST2(R3) ; Set other info
;RNG0001         51    00C8 C5    D0   1B5D    .1          MOVL    UCB$L_XQ_CDB(R5),R1     ; Get CDB address
;RNG0001                  06    13   1B62    .2          BEQL    IO_DONE2                ; Br if no CDB
;RNG0001               010C C1    C8   1B64    .3          BISL    CDB_L_DEVDEPEND(R1),IRP$L_IOST2(R3) ; Set controller bits
                         3C A3         1B68
;RNG0001                                 1B6A    .4 IO_DONE2:                            ; P1 IOST2 already setup
;RNG0001                  54    DD   1B6A    .5          PUSHL   R4                      ; Save R4
;RNG0001         54    00C8 C5    D0   1B6C    .6          MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-8                       07    E1   1B71  5460          BBC     #IRP$V_DIAGBUF,-        ; Br if no diagnotic buffer
                      22 2A A3         1B73  5461                  IRP$W_STS(R3),20$       ;
                   50    4C A3    D0   1B76  5462          MOVL    IRP$L_DIAGBUF(R3),R0    ; Get diagnostic buffer address
                   08 A0    1A    B1   1B7A  5463          CMPW    #RHDR_C_LENGTH,DIAG_W_SIZE(R0) ; Is this a real diag buffer?
                            18    13   1B7E  5464          BEQL    20$                     ; Br if not - no diagnostic info
                50    60    08    C1   1B80  5465          ADDL3   #8,(R0),R0              ; Address buffer pas* start time
                 00000000'GF    7D   1B84  5466          MOVQ    G^EXE$GQ_SYSTIME,(R0)+  ; Insert stop time
                            80         1B8A
                51    0110 C4    D0   1B8B  5467          MOVL    CDB_L_UCB0(R4),R1       ; Get address of UCB #0
                80    0082 C1    3C   1B90  5468          MOVZWL  UCB$W_ERRCNT(R1),(R0)+  ; Insert error counter
                        0218     30   1B95  5469          BSBW    REG_DUMP                ; Dump registers
                           54 8ED0   1B98  5470 20$:       POPL    R4                      ; Restore R4
                 00000000'GF    17   1B9B  5471          JMP     G^COM$POST              ; Post the I/O and return
```

```
                        1BA1  5473          .SBTTL  ASSEM_PKTS - Assemble receive packets
                        1BA1  5474 ;++
                        1BA1  5475 ; ASSEM_PKTS - Assemble receive packets
                        1BA1  5476 ;
                        1BA1  5477 ; Functional description:
                        1BA1  5478 ;
                        1BA1  5479 ; This routine assembles all receive packets into one chain of complex
;RNG0001                1BA1    .1 ; buffers.
;RNG0001                1BA1    .2 ;
;RNG0001                1BA1    .3 ;  Inputs:
;RNG0001                1BA1    .4 ;
;RNG0001                1BA1    .5 ;       R4 = CDB address
-6                      1BA1  5486 ;
                        1BA1  5487 ;       IPL = FIPL
                        1BA1  5488 ;
                        1BA1  5489 ;  Outputs:
                        1BA1  5490 ;
                        1BA1  5491 ;       R0 = Status for request
                        1BA1  5492 ;       R2 = Address of first receive buffer in chain
                        1BA1  5493 ;       R6 = Address of last buffer in receive ring
                        1BA1  5494 ;       R1,R3 are destroyed.
                        1BA1  5495 ;       All other registers are preserved.
                        1BA1  5496 ;
                        1BA1  5497 ;  Implicit Outputs:
                        1BA1  5498 ;
                        1BA1  5499 ;       IRP$V_CHAIN and IRP$V_COMPLX bits set in IRP$W_STS if the receive
                        1BA1  5500 ;       buffer is comprised of complex chained buffers.
                        1BA1  5501 ;--
                        1BA1  5502          .ENABL  LSB
                        1BA1  5503 ASSEM_PKTS::                                 ; Assemble receive packets
                0095     30 1BA1  5504          BSBW    NEXTMSG                  ; Get first message
;RNG0001           68 50  E9 1BA4    .1          BLBC    R0,100$                  ; Br if none
;RNG0001                   1BA7    .2 ;
;RNG0001                   1BA7    .3 ; Save number of messages  in chain in CXB$W_R_NCHAIN and total size of
;RNG0001                   1BA7    .4 ; all messages in CXB$W_BCNT
;RNG0001                   1BA7    .5 ;
;RNG0001          1C A2    01 B0 1BA7    .6          MOVW    #1,CXB$W_R_NCHAIN(R2)    ; Compute total number of buffers
;RNG0001          50   0C  A2 B0 1BAB    .7          MOVW    CXB$W_LENGTH(R2),R0      ; Get size of message
;RNG0001               0F  E1 1BAF    .8          BBC     #RCV_STS_V_LAST,-        ; Br if end of packet
;RNG0001             54 14 A2       1BB1    .9           CXB$W_R_STS(R2),40$     ; ..all done with this loop
;RNG0001                1A A2 B4 1BB4   .10          CLRW    CXB$W_BCNT(R2)           ; Init total size of buffers so far
-11                            52 DD 1BB7 5516          PUSHL   R2                       ; Save first receive buffer address
                         53    01 9A 1BB9 5517          MOVZBL  #MAX_C_CHAIN,R3          ; Allow n messages in chain
                           06 A6 B0 1BBC 5518 10$:       MOVW    RCV_Q_LEN(R6),-          ; Set size of buffer to maximum per rcv
                           0C A2    1BBF 5519            CXB$Q_LENGTH(R2)         ;
                              52 DD 1BC1 5520          PUSHL   R2                       ; Save address of current bufr in chain
                            0073 30 1BC3 5521          BSBW    NEXTMSG                  ; Try for next message
                              51 8ED0 1BC6 5522          POPL    R1                       ; Get address of last buffer in chain
                           21 50  E9 1BC9 5523          BLBC    R0,20$                   ; Toss all messages on error
                        50    6E  D0 1BCC 5524          MOVL    (SP),R0                  ; Get address of first in chain
                     10 A1    52  D0 1BCF 5525          MOVL    R2,CXB$L_LINK(R1)        ; Store address in chain
                           0C A1 A0 1BD3 5526          ADDW    CXB$W_LENGTH(R1),-       ; Compute total size of all buffers
                           1A A0    1BD6 5527            CXB$Q_BCNT(R0)           ;  in chain - so far
                           1C A0 B6 1BD8 5528          INCW    CXB$W_R_NCHAIN(R0)       ; Compute number of msgs in chain
                              0F  E1 1BDB 5529          BBC     #RCV_STS_V_LAST,-        ; Br if end of packet
                        0F 14 A2       1BDD 5530            CXB$W_R_STS(R2),30$      ;
                           D9 53  F5 1BE0 5531          SOBGTR  R3,10$                   ; Loop if more than two in chain
```

```
                     1BE3  5532 ;
                     1BE3  5533 ; Done with loop and LAST bit still not set - toss all messages
                     1BE3  5534 ;
                     1BE3  5535          INCC    CDB_W_OVRCTR(R4),W      ; Count as hardware error
                     1BED  5536 ;
                     1BED  5537 ; Error exit
                     1BED  5538 ;
           21   11   1BED  5539 20$:     BRB     TOSAMSG                 ; Toss all messages
                     1BEF  5540
      51   52   D0   1BEF  5541 30$:     MOVL    R2,R1                   ; Save address of last message
           52 8ED0   1BF2  5542          POPL    R2                      ; Return first message address in R2
        0B A1   90   1BF5  5543          MOVB    CXB$B_R_FLAGS(R1),-     ; Save only last message buffer info
        0B A2        1BF8  5544           CXB$B_R_FLAGS(R2)              ;  in first message of chain
        14 A1   B0   1BFA  5545          MOVW    CXB$W_R_STS(R1),-       ; DITTO
        14 A2        1BFD  5546           CXB$W_R_STS(R2)                ;
   50   0C A1   B0   1BFF  5547          MOVW    CXB$W_LENGTH(R1),R0     ; Save size of entire message
        1A A2   A2   1C03  5548          SUBW    CXB$W_BCNT(R2),-        ; Compute size of last message
        0C A1        1C06  5549           CXB$W_LENGTH(R1)               ;  and store in CXB format
1A A2   50   B0   1C08  5550 40$:     MOVW    R0,CXB$W_BCNT(R2)       ; Return size of complete message
   50   01   9A   1C0C  5551 50$:     MOVZBL  S^#SS$_NORMAL,R0        ; Return success!
                05   1C0F  5552 100$:    RSB
                     1C10  5553 ;
                     1C10  5554 ; Toss bad messages
                     1C10  5555 ;
           52 8ED0   1C10  5556 TOSAMSG:POPL    R2                      ; Restore R2
                     1C13  5557 TOSSMSG:INCC    CDB_W_LBECTR(R4),W      ; Up the counter
        10 A2   DD   1C1D  5558 110$:    PUSHL   CXB$L_LINK(R2)          ; Save address of next in chain
         F812   30   1C20  5559          BSBW    ADDRCVLIST              ; Add buffer to receive list
           52 8ED0   1C23  5560          POPL    R2                      ; Restore address of next in chain
           F5   12   1C26  5561          BNEQ    110$                    ; Br if more in chain
           50   D4   1C28  5562 120$:    CLRL    R0                      ; Assume failure
           0F   E0   1C2A  5563          BBS     #RCV_STS_V_LAST,-       ; Br if NOT end of  hain
      03 08 A6        1C2C  5564           RCV_W_STS(R6),130$             ;  get rest of message
            6F   31   1C2F  5565          BRW     ASSEM_PKTS              ; Else, try for next valid message
           05   10   1C32  5566 130$:    BSBB    NEXTMSG                 ; Get next message
        D8 50   E9   1C34  5567          BLBC    R0,100$                 ; Br if none
           EF   11   1C37  5568          BRB     120$                    ; Check if more possible
                     1C39  5569          .DSABL  LSB
                     1C39  5570 ;
                     1C39  5571 ; Find next message and check ownership
                     1C39  5572 ;
           50   D4   1C39  5573 NEXTMSG:CLRL    R0                      ; Assume failure
         0106 C4   95   1C3B  5574          TSTB    CDB_B_RCVCNT(R4)        ; Any more receives in progress?
           01   12   1C3F  5575          BNEQ    5$                      ; Br if yes
                05   1C41  5576 1$:      RSB                             ; Else, return
                     1C42  5577
   56   0104 C4   9A   1C42  5578 5$:      MOVZBL  CDB_B_LASTRCV(R4),R6    ; Get last ring entry completed inx
   56     7C A446 D0   1C47  5579          MOVL    CDB_L_RRINGVA(R4)[R6],R6; Get last ring entry address
05 08 A6   0F   E1   1C4C  5580          BBC     #RCV_STS_V_LAST,RCV_W_STS(R6),10$ ; Br if done
EB 08 A6   0E   E1   1C51  5581          BBC     #RCV_STS_V_ERR,RCV_W_STS(R6),1$ ; Br if not done
0B A6   0A A6   91   1C56  5582 10$:     CMPB    RCV_W_LENB(R6),RCV_W_LENB+1(R6) ; Are we really done?
           73   12   1C5B  5583          BNEQ    90$                     ; Br if not, leave now
   52   00F4 D4   0F   1C5D  5584          REMQUE  @CDB_Q_RCVPND(R4),R2    ; Get next receive
           6C   1D   1C62  5585          BVS     90$                     ; Br if none available (yet)
        10 A2   D4   1C64  5586          CLRL    CXB$L_LINK(R2)          ; Assume not a chained buffer
         0106 C4   97   1C67  5587          DECB    CDB_B_RCVCNT(R4)        ; One less receive pending
         0104 C4   96   1C6B  5588          INCB    CDB_B_LASTRCV(R4)       ; Bump ring pointer
```

```
                     F8 8F    8A  1C6F  5589          BICB    #^C<MAX_C_RCV-1>,-      ; Modulo receive ring entry size
                   0104 C4        1C72  5590                   CDB_B_CASTRCV(R4)       ;   ...
                                  1C75  5591          CLRBIT  #RCV_DSC_V_VALID,-      ; Indicate that buffer is not valid
                                  1C75  5592                   RCV_W_ADDRHI(R6)        ;
                                  1C7A  5593 ;
                                  1C7A  5594 ; Compute buffer size
                                  1C7A  5595 ;
                   F8FF 8F    AB  1C7A  5596          BICW3   #^C<RCV_STS_M_RLEN>,-   ; Store length <10:8>
                     08 A6        1C7E  5597                   RCV_W_STS(R6),-         ;
                     0C A2        1C80  5598                   CXB$W_LENGTH(R2)        ;
                     0A A6    90  1C82  5599          MOVB    RCV_W_LENB(R6),-        ; Store length in CXB <7:0>
                     0C A2        1C85  5600                   CXB$W_LENGTH(R2)        ;
                        2E    A0  1C87  5601          ADDW    #XQ_C_ADDRCV-<XQ_C_HEADER+XQ_C_CRC>,- ; Add in missed count
                     0C A2        1C89  5602                   CXB$W_LENGTH(R2)        ;   ..minus header and CRC
                     08 A6    B0  1C8B  5603          MOVW    RCV_W_STS(R6),-         ; Save status flags in CXB
                     14 A2        1C8E  5604                   CXB$W_R_STS(R2)
                     01 A6    90  1C90  5605          MOVB    RCV_W_FLAG+1(R6),-      ; Save flags byte (high word)
                     0B A2        1C93  5606                   CXB$B_R_FLAGS(R2)       ;
                                  1C95  5607 ;
                                  1C95  5608 ; Adjust quota and release mapping slot
                                  1C95  5609 ;
;RNG0001           02EE C4    95  1C95    .1          TSTB    CDB_B_AQUOTA(R4)        ; Are we running on extra QUOTA?
;RNG0001                 06    13  1C99    .2          BEQL    30$                     ; Br if not, replenish quota
-2                 02EE C4    97  1C9B  5612          DECB    CDB_B_AQUOTA(R4)        ; Else, decrement extra QUOTA
                        07    11  1C9F  5613          BRB     40$                     ; Continue
                                  1CA1  5614
                   0108 C4    A0  1CA1  5615 30$:     ADDW    CDB_W_BSZ(R4),-         ; Replenish CDB quota
                   010A C4        1CA5  5616                   CDB_W_QUOTA(R4)         ;
                 51   22 A2    9A  1CA8  5617 40$:     MOVZBL  CXB$B_XQ_SLOT(R2),R1    ; Get mapping slot number used
                                  1CAC  5618          CLRBIT  R1,CDB_B_RCVMAP(R4)     ; Clear in use flag
                                  1CB1  5619          CPUDISP <<790,80$>,-
                                  1CB1  5620                   <780,80$>,-
                                  1CB1  5621                   <750,80$>,-
                                  1CB1  5622                   <730,80$>,-
;RNG0001                          1CB1    .1                 <UV2,80$>,-
;RNG0001                          1CB1    .2                 <UV1,100$>>             ; Copy data on MicroVAX I
;RNG0001                          1CCD    .3
;RNG0001                          1CCD    .4 80$:     ;  All but MicroVAX I
;RNG0001                          1CCD    .5
;RNG0001            50    01    9A  1CCD    .6          MOVZBL  S^#SS$_NORMAL,R0        ; Return success!
;RNG0001                  05    1C  1CD0    .7 90$:     RSB
;RNG0001                          1CD1    .8
;RNG0001                          1CD1    .9 100$:    ;**********************
;RNG0001                          1CD1   .10          ; For MicroVAX I, ONLY.
;RNG00C1                          1CD1   .11          ;**********************
;RNG0001                          1CD1   .12
;RNG0001                    0E    E0  1CD1   .13          BBS     #RCV_STS_V_ERR,-        ; Br if this packet is in error
;RNG0001            F7 14 A2        1CD3   .14                   CXB$W_R_STS(R2),80$
;RNG0001               0C A2    B1  1CD6   .15          CMPW    CXB$W_LENGTH(R2),CDB_W_BSZ(R4)  ; Check size of received data
                    0108 C4        1CD9
;RNG0001                    16    1A  1CDC   .16          BGTRU   110$                    ; Br if bad size
;RNG0001          50   00C4 C441    D0  1CDE   .17          MOVL    CDB_L_RCV_VA(R4)[R1],R0 ; Else, get address of contiguous buffer
;RNG0001                    0E    A1  1CE4   .18          ADDW3   #XQ_C_HEADER,-          ; Add back in the header
;RNG0001              53    0C A2    1CE6   .19                   CXB$W_LENGTH(R2),R3     ;
;RNG0001                    34    BB  1CE9   .20          PUSHR   #^M<R2,R4,R5>           ; Save registers
;RNG0001                60    53    28  1CEB   .21          MOVC3   R3,(R0),CXB$T_R_DATA(R2) ; Move the data
                         38 A2        1CEE
```

```
;RNG0001                34   BA  1CF0   .22          POPR    #^M<R2,R4,R5>           ; Restore registers
;RNG0001                D9   11  1CF2   .23          BRB     80$                     ; Continue
;RNG0001                        1CF4   .24
;RNG0001                05   E0  1CF4   .25 110$:    BBS     #CDB_STS_V_SETUP,-      ; Br if this is a SETUP packet
;RNG0001          D3 0246 C4         1CF6   .26                CDB_B_STS(R4),80$       ;  (no error)
;RNG0001                        1CFA   .27          SETBIT  #RCV_STS_V_ERR,-        ; Else, indicate error
;RNG0001                        1CFA   .28                CXB$W_R_STS(R2)         ;  and let it get tossed
-23                     CC   11  1CFF  5646         BRB     80$                     ; Continue
```

```
                                1D01  5648          .SBTTL  MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
                                1D01  5649 ;++
                                1D01  5650 ; MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
                                1D01  5651 ;
                                1D01  5652 ; FUNCTIONAL DESCRIPTION:
                                1D01  5653 ;
                                1D01  5654 ; This routine is called to process a remote request to read the LINE counters.
                                1D01  5655 ;
                                1D01  5656 ; Inputs:
                                1D01  5657 ;
                                1D01  5658 ;       R2 = Address of the received message
                                1D01  5659 ;       R4 = CDB address
                                1D01  5660 ;
                                1D01  5661 ; Outputs:
                                1D01  5662 ;
                                1D01  5663 ;       R0,R1,R2,R3,R5 are destroyed
                                1D01  5664 ;       R4 is preserved
                                1D01  5665 ;--
                                1D01  5666
                                1D01  5667 MOP_CTR_REQUEST::                    ; Process a read counters request
              10 A2     D5      1D01  5668          TSTL    CXB$L_LINK(R2)      ; Is this a chained message?
                 24     12      1D04  5669          BNEQ    80$                 ; Br if yes, return packets
                        B1      1D06  5670          CMPW    #IRP$C_LENGTH-      ; Is a receive buffer large enough?
                                1D07  5671                  +CXB$C_HEADER+3-
                                1D07  5672                  +MOP_CTR_SIZE+8,-
            0168 8F             1D07  5673                  CXB$W_SIZE(R2)
              08 A2             1D0A
                 1C     1A      1D0C  5674          BGTRU   80$                 ; Br if no, ignore message
        55    0110 C4     D0    1D0E  5675          MOVL    CDB_L_UCB0(R4),R5   ; Get address of UCB #0
          50     3E A2    7D    1D13  5676          MOVQ    CXB$G_R_SRC(R2),R0  ; Save source node address
                 47 A2    B0    1D17  5677          MOVW    CXB$T_R_USERDAT+1(R2),-; Copy the request ID
              010D C2           1D1A  5678                  IRP$C_LENGTH+CXB$C_HEADER+1(R2) ;  to the Message block
                0555     30     1D1D  5679          BSBW    BLD_IRP             ; Turn the message block into an IRP
        40 A3    50     7D     1D20  5680          MOVQ    R0,IRP$Q_STATION(R3) ; Set the return node address
        14 A3    54     D0     1D24  5681          MOVL    R4,IRP$L_ASTPRM(R3) ; Save the CDB address
                 0C     11     1D28  5682          BRB     MOP_CTR_BUILD       ; Build the response message
                                1D2A  5683 ;
                                1D2A  5684 ; Error - returns packets to receive queue
                                1D2A  5685 ;
              10 A2     DD      1D2A  5686 80$:     PUSHL   CXB$L_LINK(R2)      ; Save next in chain
               F705     30      1D2D  5687          BSBW    ADDRCVLIST          ; Add buffer to receive list
                  52 8ED0       1D30  5688          POPL    R2                  ; Restore next buffer
                  F5    12      1D33  5689          BNEQ    80$                 ; Loop if more
                        05      1D35  5690          RSB                         ; Return to caller
                                1D36  5691
```

```
                     1D36  5693          .SBTTL  MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
                     1D36  5694 ;++
                     1D36  5695 ; MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
                     1D36  5696 ;
                     1D36  5697 ; FUNCTIONAL DESCRIPTION:
                     1D36  5698 ;
                     1D36  5699 ; This routine is called to build the return message in response to a
                     1D36  5700 ; MOP read counters request.
                     1D36  5701 ;
                     1D36  5702 ; Inputs:
                     1D36  5703 ;
                     1D36  5704 ;       R3 = IRP address
                     1D36  5705 ;       R4 = CDB address
                     1D3?  5706 ;       R5 = UCB address of UNIT 0
                     1D:   5707 ;
                     1D36  5708 ; Outputs:
                     1D36  5709 ;
                     1D36  5710 ;       R4 = CDB address
                     1D36  5711 ;       R5 = UCb address of UNIT 0
                     1D36  5712 ;
                     1D36  5713 ;       R0,R1 are destroyed
                     1D36  5714 ;--
                     1D36  5715
                     1D36  5716 MOP_CTR_BUILD:                          ; Build the MOP counter return msg
                     1D36  5717
     01F8 8F    BB   1D36  5718          PUSHR   #^M<R3,R4,R5,R6,R7,R8>  ; Save registers
     56   54    D0   1D3A  5719          MOVL    R4,R6                   ; Copy CDB address
 53  010C C3    9E   1D3D  5720          MOVAB   IRP$C_LENGTH+-          ; Point to start of message
                     1D42  5721                  CXB$C_HEADER(R3),R3     ;    block
     83   0B    90   1D42  5722          MOVB    #NI_CTR_REPLY,(R3)+     ; Set function to reply
          83    B5   1D45  5723          TSTW    (R3)+                   ; Skip request ID (filled in earlier)
 57  E426 CF    9E   1D47  5724          MOVAB   MOPCTRTAB,R7            ; Get address of MOP counters
     58   87    3C   1D4C  5725  20$:    MOVZWL  (R7)+,R8                ; Get the offset to the counter desired
          19    13   1D4F  5726          BEQL    30$                     ; Br if end of table
     58   56    C0   1D51  5727          ADDL    R6,R8                   ; Compute address of the counter
 67  07   00    EF   1D54  5728          EXTZV   #0,#7,(R7),R2           ; Get the width field without bitmap
          52         1D58
 63  68   52    28   1D59  5729          MOVC3   R2,(R8),(R3)            ; Copy the counter
     51   87    9A   1D5D  5730          MOVZBL  (R7)+,R1                ; Get the width field again
   E8 51   07    E1   1D60  5731          BBC     #7,R1,20$               ; Br if no BITMAP to return
   83   FE  A8    B0   1D64  5732          MOVW    -2(R8),(R3)+            ; Else, store the BITMAP
          E2    11   1D68  5733          BRB     20$                     ; Loop for next counter
                     1D6A  5734
     01F8 8F    BA   1D6A  5735  30$:    POPR    #^M<R3,R4,R5,R6,R7,R8>  ; Restore registers
                     1D6E  5736
     226C'CF    9E   1D6E  5737          MOVAB   W^DELETE_BLOCK,IRP$L_PID(R3) ; Set the return address
       0C A3         1D72
 52  00C4 C3    9E   1D74  5738          MOVAB   IRP$C_LENGTH(R3),R2     ; Get address of CXB
   0A A2    1B    90   1D79  5739          MOVB    #DYN$C_CXB,CXB$B_TYPE(R2) ; Make it look like a CXB
     005C 8F    B0   1D7D  5740          MOVW    #MOP_CTR_SIZE+8+3,-     ; Set size of transfer
       1A A2         1D81  5741                  CXB$Q_BCNT(R2)          ;
                     1D83  5742          ASSUME  CXB$C_HEADER  EQ  CXB$T_DATA+XQ_C_HEADER
          3A   B0    1D83  5743          MOVW    #CXB$T_DATA,-           ; Set offset to start of data
       18 A2         1D85  5744                  CXB$W_BOFF(R2)
   24 A2    53    D0   1D87  5745          MOVL    R3,CXB$L_T_IRP(R2)      ; Save IRP address in CXB
       40 A3    7D   1D8B  5746          MOVQ    IRP$Q_STATION(R3),-     ; Set STATION in CXB
       28 A2         1D8E  5747                  CXB$Q_STATION(R2)
```

```
                    01   90   1D90  5748          MOVB    #XQ_FC_V_XMIT,-          ; Set function request in CXB
                 20 A2         1D92  5749                  CXB$B_XQ_FUNC(R2)
55               1C A3   D0   1D94  5750          MOVL    IRP$L_UCB(R3),R5         ; Get the UCB address
52               3A A2   9E   1D98  5751          MOVAB   CXB$T_DATA(R2),R2        ; Set R2 to start of data
                 40 A3   7D   1D9C  5752          MOVQ    IRP$Q_STATION(R3),-      ; Set th destination address
                    62        1D9F  5753                  XBUF_G_DEST(R2)
              0260 8F   B0   1DA0  5754          MOVW    #NI_CTR_PROTYP,-         ; Store the protocol type
                 0C A2         1DA4  5755                  XBUF_W_TYPE(R2)
              00C4 C3   0E   1DA6  5756          INSQUE  IRP$C_LENGTH(R3),-       ; Insert request on request queue
              00F0 D4         1DAA  5757                  @CDB_G_XMTREQ+4(R4)
                 E6D7   31   1DAD  5758          BRW     XMT_ALT_START            ; Startup the reply
                             1DB0  5759
```

```
                                                H  1

                     1DB0  5761          .SBTTL  REG_DUMP - DEQNA ERROR LOG AND DIAGNOSTICS REGISTER DUMP
                     1DB0  5762 ;++
                     1DB0  5763 ; REG_DUMP - DIAGNOSTICS REGISTER DUMP ROUTINE
                     1DB0  5764 ;
                     1DB0  5765 ; FUNCTIONAL DESCRIPTION:
                     1DB0  5766 ;
                     1DB0  5767 ; This routine is used to return the DEQNA error log and diagnostics
                     1DB0  5768 ; buffer on error or diagnostic QIO function.
                     1DB0  5769 ;
                     1DB0  5770 ; Inputs:
                     1DB0  5771 ;
                     1DB0  5772 ;       R0 = Address of the buffer @ DIAG_L_EXTRA
                     1DB0  5773 ;       R4 = CDB address
                     1DB0  5774 ;       R5 = UCB address of the unit
                     1DB0  5775 ;
                     1DB0  5776 ; Outputs:
                     1DB0  5777 ;
                     1DB0  5778 ;       R0,R1 are destroyed
                     1DB0  5779 ;       R4,R5 are preserved
                     1DB0  5780 ;--
                     1DB0  5781 REG_DUMP::
                     1DB0  5782          ASSUME  DIAG_L_DEPEND EQ DIAG_L_EXTRA+4
                     1DB0  5783          ASSUME  DIAG_W_CSR EQ DIAG_L_DEPEND
                     1DB0  5784          ASSUME  DIAG_W_ERR EQ DIAG_W_CSR+2
                     1DB0  5785          ASSUME  DIAG_W_ERR2 EQ DIAG_W_ERR+2
                     1DB0  5786          ASSUME  DIAG_G_HWA EQ DIAG_W_ERR2+2
            80   06  9A  1DB0  5787          MOVZBL  #DIAG_C_EXTRA,(R0)+     ; Insert number of returned long words
:RNG0001      80   10  A4  B0  1DB3    .1          MOVW    CDB_L_LASTCSR(R4),(R0)+ ; Insert last CSR contents
-1          80   0119  C4  9B  1DB7  5789          MOVZBW  CDB_B_DIAG1(R4),(R0)+   ; Return error flags info
            80   011A  C4  B0  1DBC  5790          MOVW    CDB_W_DIAG2(R4),(R0)+   ; Return extra error info
            80   0250  C4  D0  1DC1  5791          MOVL    CDB_G_HWA(R4),(R0)+     ; Return hardware physical address
            80   0254  C4  B0  1DC6  5792          MOVW    CDB_G_HWA+4(R4),(R0)+   ;       ...
                         05  1DCB  5793          RSB
```

```
                     1DCC  5795          .SBTTL  RESTART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
                     1DCC  5796 ;++
                     1DCC  5797 ; RESTART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
                     1DCC  5798 ;
                     1DCC  5799 ; Functional description:
                     1DCC  5800 ;
                     1DCC  5801 ; This routine is entered when the RESTART delta time has expired. The action
                     1DCC  5802 ; is to check if the specified unit has been restarted, if so, then it is
                     1DCC  5803 ; automatically restarted.
                     1DCC  5804 ;
                     1DCC  5805 ; Inputs:
                     1DCC  5806 ;
                     1DCC  5807 ;       R4 = CDB address
                     1DCC  5808 ;       R5 = TQE address (but must be at least as long as an IRP)
                     1DCC  5809 ;
                     1DCC  5810 ;       IPL = IPL$_TIMEP
                     1DCC  5811 ;
                     1DCC  5812 ; Implicit inputs:
                     1DCC  5813 ;
                     1DCC  5814 ;       IRP$L_RBOFF(R5) = UCB address for UNIT
                     1DCC  5815 ;
                     1DCC  5816 ; Outputs:
                     1DCC  5817 ;
                     1DCC  5818 ;       R0-R3 are destroyed.
                     1DCC  5819 ;--
                     1DCC  5820          .ENABL  LSB
                     1DCC  5821 RESTART_ROUT::                          ; Process expiration of restart timer
          55   DD    1DCC  5822          PUSHL   R5                     ; Save R5
                     1DCE  5823          ASSUME  IRP$L_RBOFF GT TQE$C_LENGTH
     52   30 A5   9E  1DCE  5824          MOVAB   TQE$C_LENGTH(R5),R2    ; Point to IRP portion of TQE
55   0098 C2   D0  1DD2  5825          MOVL    IRP$L_RBOFF(R2),R5     ; Reset R5 to UCB address
                     1DD7  5826 ;
                     1DD7  5827 ; Turn TQE into an IRP
                     1DD7  5828 ;
          53   82   7E  1DD7  5829          MOVAQ   (R2)+,R3               ; Copy IRP address, skip to size field
                     1DDA  5830          ASSUME  IRP$W_SIZE EQ 8
                     1DDA  5831          ASSUME  IRP$B_TYPE EQ IRP$W_SIZE+2
                     1DDA  5832          ASSUME  IRP$B_RMOD EQ IRP$B_TYPE+1
          52   02   C0  1DDA  5833          ADDL    #2,R2                  ; Skip size field
          82   0A   B0  1DDD  5834          MOVW    #DYN$C_IRP,(R2)+       ; Set type to IRP
                     1DE0  5835          ASSUME  IRP$L_PID EQ IRP$B_RMOD+1
82   1E25'CF   9E  1DE0  5836          MOVAB   W^RETURN_IRP,(R2)+     ; Set return address form IOPOST
                     1DE5  5837          ASSUME  IRP$L_AST EQ IRP$L_PID+4
                     1DE5  5838          ASSUME  IRP$L_ASTPRM EQ IRP$L_AST+4
               82   7C  1DE5  5839          CLRQ    (R2)+                  ; Clear AST, ASTPRM
                     1DE7  5840          ASSUME  IRP$L_WIND EQ IRP$L_ASTPRM+4
                     1DE7  5841          ASSUME  IRP$L_UCB EQ IRP$L_WIND+4
               82   D4  1DE7  5842          CLRL    (R2)+                  ; Clear WIND
          82   55   D0  1DE9  5843          MOVL    R5,(R2)+               ; Set UCB address
                     1DEC  5844          ASSUME  IRP$W_FUNC EQ IRP$L_UCB+4
                     1DEC  5845          ASSUME  IRP$B_XQ_FUNC EQ IRP$W_FUNC+1
                     1DEC  5846          ASSUME  IRP$B_EFN EQ IRP$W_FUNC+2
                     1DEC  5847          ASSUME  IRP$B_PRI EQ IRP$B_EFN+1
                     1DEC  5848          ASSUME  IRP$L_IOSB EQ IRP$B_PRI+1
82   0500 8F   B0  1DEC  5849          MOVW    #<XQ_FC_V_RESTART@8>,(R2)+ ; Set function request
               82   B4  1DF1  5850          CLRW    (R2)+                  ; Clear EFN, PRI
               82   D4  1DF3  5851          CLRL    (R2)+                  ; Clear IOSB
```

```
                     1DF5  5852          ASSUME  IRP$W_CHAN EQ IRP$L_IOSB+4
                     1DF5  5853          ASSUME  IRP$W_STS EQ IRP$W_CHAN+2
                     1DF5  5854          ASSUME  IRP$L_SVAPTE EQ IRP$W_STS+2
          82   7C    1DF5  5855          CLRQ    (R2)+                   ; Clear CHAN, STS, SVAPTE
                     1DF7  5856          ASSUME  IRP$W_BOFF EQ IRP$L_SVAPTE+4
                     1DF7  5857          ASSUME  IRP$W_BCNT EQ IRP$W_BOFF+2
                     1DF7  5858          ASSUME  IRP$W_BCNT EQ IRP$L_BCNT
                     1DF7  5859          ASSUME  IRP$L_MEDIA EQ IRP$W_BCNT+6
          82   7C    1DF7  5860          CLRQ    (R2)+                   ; Clear BOFF, BCNT
    82    00   3C    1DF9  5861          MOVZWL  #<XQ_FC_V_INIT@8>,(R2)+ ; Set MEDIA
          82   D4    1DFC  5862          CLRL    (R2)+                   ; Clear MEDIA+4
                     1DFE  5863 ;
                     1DFE  5864 ; RESTART the UNIT
                     1DFE  5865 ;
                     1DFE  5866          DSBINT  UCB$B_FIPL(R5)          ; Raise IPL
  4000 8F    AA    1E05  5867          BICW    #UCB$M_XQ_INTERLOCK,-   ; Clear the RESTART interlock
    68 A5          1E09  5868                  UCB$W_DEVSTS(R5)        ;
       00   E2    1E0B  5869          BBSS    #UCB$V_XQ_INITED,-      ; Br if unit already inited
 09 68 A5          1E0D  5870                  UCB$W_DEVSTS(R5),10$    ;
     F0A2   30    1E10  5871          BSBW    START                   ; Start protocol
    08 50   E8    1E13  5872          BLBS    R0,30$                  ; Br if success
                  1E16  5873
     F085   30    1E16  5874          BSBW    STOP                    ; Shutdown unit
 55    53   D0    1E19  5875 10$:     MOVL    R3,R5                   ; Point R5 to IRP
       07   10    1E1C  5876          BSBB    RETURN_IRP              ; Return the IRP
                  1E1E  5877 30$:     ENBINT                          ; Re-enable interrupts
       55 8ED0    1E21  5878          POPL    R5                      ; Restore R5
          05      1E24  5879          RSB                             ; Return to caller
                  1E25  5880
                  1E25  5881 RETURN_IRP:
 50    D0 A5   9E    1E25  5882          MOVAB   -TQE$C_LENGTH(R5),R0    ; Get address of start of structure
00000000'GF   17    1E29  5883          JMP     G^COM$DRVDEALMEM        ; Deallocate the IRP
```

```
                                                 K 1
                     1E2F  5885          .SBTTL  TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
                     1E2F  5886 ;++
                     1E2F  5887 ; TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
                     1E2F  5888 ;
                     1E2F  5889 ; Functional description:
                     1E2F  5890 ;
                     1E2F  5891 ; This routine is entered when the TQE delta time has expired. The action is to
                     1E2F  5892 ; check all timer cells and shut down the controller if any have expired.
                     1E2F  5893 ;
                     1E2F  5894 ; Inputs:
                     1E2F  5895 ;
                     1E2F  5896 ;       R4 = CDB address
                     1E2F  5897 ;       R5 = TQE address
                     1E2F  5898 ;
                     1E2F  5899 ;       IPL = IPL$_TIMER
                     1E2F  5900 ;
                     1E2F  5901 ; Outputs:
                     1E2F  5902 ;
                     1E2F  5903 ;       R0-R3 are destroyed.
                     1E2F  5904 ;       R4,R5 are preserved
                     1E2F  5905 ;--
                     1E2F  5906          .ENABL  LSB
                     1E2F  5907 TQE_TIMER::                              ; Process expiration of TQE timer
          50  0110 C4  D0  1E2F  5908          MOVL    CDB_L_UCBO(R4),R0       ; Get first UCB address
                     1E34  5909          DSBINT  UCB$B_DIPL(R0)          ; Sync access to UCB and CDB
;RNG0001                 1E3B    .1          ASSUME  CDB_STS_V_INITED  EQ  0
;RNG0001    28 0246 C4   E9  1E3B    .2          BLBC    CDB_B_STS(R4),STOP_TQE  ; Br if NOT inited
;RNG0001                 1E40    .3
;RNG0001                 1E40   .12 10$:     ;
;RNG0001                 1E40   .13          ; Check transmit timeout timer
;RNG0001                 1E40   .14          ;
;RNG0001       020A C4   95  1E40   .15          TSTB    CDB_B_TIM_XMT(R4)       ; Is the xmit timer going?
;RNG0001            06   13  1E44   .16          BEQL    20$                     ; Br if not
;RNG0001       020A C4   97  1E46   .17          DECB    CDB_B_TIM_XMT(R4)       ; Timer expired?
;RNG0001            11   13  1E4A   .18          BEQL    TIMOUT                  ; Br if yes
;RNG0001            0C   E1  1E4C   .19 20$:     BBC     #XM$V_STS_BUFFAIL,-     ; Br if NO buffer allocation failures
-7          07 010C C4          1E4E  5917                  CDB_L_DEVDEPEND(R4),30$ ;
                           1E52  5918          SETIPL  CDB_B_FIPL(R4)          ; Else, sync access to UCB & CDB
                 F5DA  30  1E56  5919          BSBW    FILRCVLIST              ; And try to replenish receive buffers
                           1E59  5920 30$:     ENBINT                          ; Restore IPL
                           1E5C  5921
-1                         1E5C  5945
                     05    1E5C  5946          RSB                             ; Return to caller
                           1E5D  5947
                           1E5D  5948 TIMOUT:  PUSHQ   R4                      ; Save R4, R5
          55   50   D0    1E60  5949          MOVL    R0,R5                   ; Copy address of UCB0
               26   10    1E63  5950          BSBB    DEV_TIMEOUT             ; Else, timeout has occurred
                           1E65  5951          POPQ    R4                      ; Restore R4, R5
                           1E68  5952
                           1E68  5953 STOP_TQE:                                ; Stop the TQE
       0B A5   04   8A     1E68  5954          BICB    #TQE$M_REPEAT,TQE$B_RQTYPE(R5) ; Stop the timer
               08   8A     1E6C  5955          BICB    #CDB_STS_M_TIMER,-      ; Indicate that timer is stopped
          0246 C4           1E6E  5956                  CDB_B_STS(R4)           ;
               04   8A     1E71  5957          BICB    #DPT$M_NOUNLOAD,-       ; OKAY to unload the driver now
 0000000D'EF               1E73  5958                  DPT$TAB+DPT_B_FLAGS     ;
               DF   11     1E78  5959          BRB     30$                     ; Leave
                           1E7A  5960          .DSABL  LSB
```

```
                                1E7A  5962          .SBTTL  TIMEOUT - TIMEOUT SERVICE ROUTINE
                                1E7A  5963 ;++
                                1E7A  5964 ; TIMEOUT - TIMEOUT SERVICE ROUTINE
                                1E7A  5965 ;
                                1E7A  5966 ; Functional description:
                                1E7A  5967 ;
                                1E7A  5968 ; This routine is entered on device timeout. The action is to
                                1E7A  5969 ; shut the unit down.
                                1E7A  5970 ;
                                1E7A  5971 ; Inputs:
                                1E7A  5972 ;
                                1E7A  5973 ;       R5 = UCB ADDRESS
                                1E7A  5974 ;
                                1E7A  5975 ;       IPL = DIPL
                                1E7A  5976 ;
                                1E7A  5977 ; Outputs:
                                1E7A  5978 ;
                                1E7A  5979 ;       R3,R4 are destroyed.
                                1E7A  5980 ;       R5 is preserved
                                1E7A  5981 ;--
                                1E7A  5982          .ENABL  LSB
                                1E7A  5983 TIMEOUT::                                ; Timeout or powerfail
          64 A5   03    AA      1E7A  5984          BICW    #UCB$M_TIM!UCB$M_INT,UCB$W_STS(R5) ; Disable timer
                                1E7E  5985
                                1E7E  5986          ASSUME  UCB$V_XQ_INITED  EQ  0
       30 68 A5    E9          1E7E  5987          BLBC    UCB$W_DEVSTS(R5),20$        ; Br if not inited
    53    01    11    78      1E82  5988          ASHL    #XQ_SOFT_V_POWER+16,#1,R3 ; Assume powerfail
 1A 64 A5    05    E0          1E86  5989          BBS     #UCB$V_POWER,UCB$W_STS(R5),10$ ; Br if powerfail
                                1E8B  5990 DEV_TIMEOUT::                            ; Hardware device timeout
;RNG0001  54   00C8 C5    D0  1E8B    .1          MOVL    UCB$L_XQ_CDB(R5),R4       ; Get CDB address
;RNG0001        20    13    1E90    .2          BEQL    20$                       ; Br if no CDB
;RNG0001  53   0114 C4    D0  1E92    .3          MOVL    CDB_L_CSR(R4),R3          ; Get CSR address
;RNG0001     0E A3    02    B0  1E97    .4          MOVW    #XQ_CSR_M_RESET,CSR(R3) ; Stop the device
-6       53   01    10    78  1E9B  5997          ASHL    #XQ_SOFT_V_TIMEOUT+16,#1,R3 ; Indicate timeout
                                1E9F  5998          SETBIT  #XM$V_STS_TIMO,CDB_L_DEVDEPEND(R4) ; Set error status
         53   4000 8F    B0  1EA5  5999 10$:       MOVW    #XQ_CSR_M_ERR,R3          ; Indicate fatal error
                   55    DD  1EAA  6000          PUSHL   R5                        ; Save UCB address
                      F797    30  1EAC  6001          BSBW    SCHED_FORK                ; Schedule a fork process
                   55 8ED0    1EAF  6002          POPL    R5                        ; Restore UCB address
                         05    1EB2  6003 20$:       RSB                               ; Return to caller
                               1EB3  6004          .DSABL  LSB
```

```
                                                M 1
                     1EB3  6006          .SBTTL  ALLOC_CDB - ALLOCATE THE CDB
                     1EB3  6007 ;++
                     1EB3  6008 ; ALLOC_CDB - ALLOCATE THE CDB
                     1EB3  6009 ;
                     1EB3  6010 ; Functional description:
                     1EB3  6011 ;
                     1EB3  6012 ; This routine allocates and initializes the CDB.
                     1EB3  6013 ;
                     1EB3  6014 ; Inputs:
                     1EB3  6015 ;
                     1EB3  6016 ;       R5 = UCB address
                     1EB3  6017 ;
                     1EB3  6018 ; Outputs:
                     1EB3  6019 ;
                     1EB3  6020 ;       R0 = Status return for request
                     1EB3  6021 ;
                     1EB3  6022 ;       All other registers are preserved.
                     1EB3  6023 ;
                     1EB3  6024 ;--
                     1EB3  6025
                     1EB3  6026 ALLOC_CDB:                              ; Allocate a CDB
           3E   BB   1EB3  6027          PUSHR   #^M<R1,R2,R3,R4,R5>     ; Save registers
51   02F0 8F   3C   1EB5  6028          MOVZWL  #CDB_C_LENGTH,R1        ; Get size of CDB allocation
 00000000'GF   16   1EBA  6029          JSB     G^EXE$ALONONPAGED       ; Try to allocate CDB
         47 50   E9   1EC0  6030          BLBC    R0,90$                  ; Br if error
                     1EC3  6031
   54    24 A5   D0   1EC3  6032          MOVL    UCB$L_CRB(R5),R4        ; Get CRB address
   10 A4    52   D0   1EC7  6033          MOVL    R2,CRB$L_AUXSTRUC(R4)   ; Store CDB address
   54    28 A5   D0   1ECB  6034          MOVL    UCB$L_DDB(R5),R4        ; Get DDB address
   54    04 A4   D0   1ECF  6035          MOVL    DDB$L_UCB(R4),R4        ; Get UCB0 address
      53    54   D0   1ED3  6036          MOVL    R4,R3                   ; Save UCB0 address
;RNG0001  00C8 C4    52   D0   1ED6     .1 10$:   MOVL    R2,UCB$L_XQ_CDB(R4)     ; Save CDB address in UCB0
;RNG0001       0250 C2    9E   1EDB     .2        MOVAB   CDB_G_HWA(R2),-         ; Store address of NI device's
-1             0090 C4         1EDF  6038                  UCB$L_NI_HWAPTR(R4)     ; unique hardware address
   54    30 A4   D0   1EE2  6039          MOVL    UCB$L_LINK(R4),R4       ; Position to next UCB
         EE   12   1EE6  6040          BNEQ    10$                     ; Continue if more UCBs
                     1EE8  6041
                     1EE8  6042 ;
                     1EE8  6043 ; Initialize CDB
                     1EE8  6044 ;
                     1EE8  6045 ;   The PADDING MODE and the ECHO MODE of the QNA will default to
                     1EE8  6046 ;   the enabled (ON) state.
                     1EE8  6047 ;
                     1EE8  6048          ASSUME  NMA$C_STATE_ON EQ 0
                     1EE8  6049
                     1EE8  6050          PUSHQ   R2                      ; Save CDB, UCB0 address
00   62   00   2C   1EEB  6051          MOVC5   #0,(R2),#0,R1,(R2)      ; Zero the structure
     62   51         1EEF
                     1EF1  6052          POPQ    R2                      ; Restore CDB, UCB0 address
                     1EF4  6053
0110 C2   53   D0   1EF4  6054          MOVL    R3,CDB_L_UCB0(R2)       ; Save address of UCB0
                     1EF9  6055 ;
                     1EF9  6056 ;  Init CDB fork block.
                     1EF9  6057 ;
                     1EF9  6058          ASSUME  CDB_L_FQFL EQ 0
                     1EF9  6059          ASSUME  CDB_L_FQBL EQ CDB_L_FQFL+4
           82   7C   1EF9  6060          CLRQ    (R2)+                   ; Skip link pointers
```

```
                                 1EFB  6061          ASSUME  CDB_W_SIZE EQ CDB_L_FQBL+4
                                 1EFB  6062          ASSUME  CDB_B_TYPE EQ CDB_W_SIZE+2
                                 1EFB  6063          ASSUME  CDB_B_FIPL EQ CDB_B_TYPE+1
          083302F0 8F    DD      1EFB  6064          MOVL    #<<^IPL$_XQ_FIPL@8>!DYN$C_CDB>@16>!- ; Set structure type and FIPL
                   82            1F01
                                 1F02  6065                  CDB_C_LENGTH,(R2)+      ;  and size
                                 1F02  6066          ASSUME  CDB_L_FPC EQ CDB_B_FIPL+1
                82   F75D CF    9E    1F02  6067          MOVAB   FORK_PROC,(R2)+         ; Set fork process address
                     50   01    9A    1F07  6068          MOVZBL  S^#SS$_NORMAL,R0        ; Return success
                          3E    BA    1F0A  6069 90$:     POPR    #^M<R1,R2,R3,R4,R5>     ; Restore registers
                                05    1F0C  6070          RSB                             ; Return to caller
```

```
                                1F0D  6072          .SBTTL  SHUTDOWN_QNA - SHUTDOWN QNA AND ALL UNITS
                                1F0D  6073 ;++
                                1F0D  6074 ; SHUTDOWN_QNA - SHUTDOWN QNA AND ALL UNITS
                                1F0D  6075 ;
                                1F0D  6076 ; Inputs:
                                1F0D  6077 ;
                                1F0D  6078 ;       R4 = CDB address
                                1F0D  6079 ;       R5 = UCB address of unit #0
                                1F0D  6080 ;
                                1F0D  6081 ;
                                1F0D  6082 ;       IPL = FIPL
                                1F0D  6083 ;
                                1F0D  6084 ; Outputs:
                                1F0D  6085 ;
                                1F0D  6086 ;       R3-R5 are preserved.
                                1F0D  6087 ;
                                1F0D  6088 ;       R0-R2 are destroyed.
                                1F0D  6089 ;--
                                1F0D  6090
                                1F0D  6091 SHUTDOWN_QNA::                          ; Shutdown QNA
:RNG0001                        1F0D    .1         ASSUME  CDB_STS_V_INITED  EQ  0
:RNG0001        01 0246 C4  E8  1F0D    .2         BLBS    CDB_B_STS(R4),10$          ; Br if QNA inited
-2                          05  1F12  6094         RSB                                ; Else, return
                            1F13  6095
                00E8 8F     BB  1F13  6096 10$:    PUSHR   #^M<R3,R5,R6,R7>           ; Save registers
                            1F17  6097 ;
                            1F17  6098 ; Shutdown QNA and reset controller status
                            1F17  6099 ;
:RNG0001        52  0114 C4 D0  1F17    .1         MOVL    CDB_L_CSR(R4),R2           ; Get CSR address
-3                          1F1C  6103         DSBINT  UCB$B_DIPL(R5)             ; Raise IPL for master clear
              0E A2   02    A8  1F23  6104         BISW    #XQ_CSR_M_RESET,CSR(R2) ; Disable device
                 F3 8F    8A  1F27  6105         BICB    #^C<CDB_STS_M_FORK_PEND!- ; Reset all but needed bits
                0246 C4         1F2A
                            1F2D  6106                     CDB_STS_M_TIMER>,CDB_B_STS(R4) ;
                            1F2D  6107         ENBINT                             ; Return to fork level
                            1F30  6108 ;
                            1F30  6109 ; Release the receive and transmit buffer map registers
                            1F30  6110 ;
                      57    D4  1F30  6111         CLRL    R7                         ; Init slot number
                            1F32  6112         ASSUME  CDB_L_RCVMAP+<4*<MAX_C_RCV-1>> EQ CDB_L_XMTMAP
:RNG0001        56    1C A4 9E  1F32    .1         MOVAB   CDB_L_RCVMAP(R4),R6        ; Get address of mapping slots
-1              53    24 A5 D0  1F36  6114 20$:    MOVL    UCB$L_CRB(R5),R3           ; Get CRB address
                            1F3A  6115         ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                            1F3A  6116         ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
                34 A3    86 D0  1F3A  6117         MOVL    (R6)+,CRB$L_INTD+VEC$W_MAPREG(R3) ; Set mapping information
                      1B 19  1F3E  6118         BLSS    30$                        ; Br if none allocated
                   37 A3 95  1F40  6119         TSTB    CRB$L_INTD+VEC$B_DATAPATH(R3) ; Is there a datapath?
                      0C 13  1F43  6120         BEQL    25$                        ; Br if not - don't do purge or release
                            1F45  6121         PURDPR                             ; Purge the data path
                            1F4B  6122         RELDPR                             ; Release the data path
                            1F51  6123 25$:    RELMPR                             ; Release the map register
                FC A6   01  CE  1F57  6124         MNEGL   #1,-4(R6)                  ; Reset mapping info
                            1F5B  6125         ASSUME  MAX_C_RCV EQ 8
                            1F5B  6126 30$:    CLRBIT  R7,CDB_B_RCVMAP(R4)        ; Clear mapping slot flag
:RNG0001        D2  57    0A  F2  1F60    .1         AOBLSS  #<MAX_C_RCV-1>+<MAX_C_XMT-1>,R7,20$ ; Loop if more map registers
-1                          1F64  6128 ;
                            1F64  6129 ; Release the PCBB map registers
```

```
                        1F64  6130 ;
      53   24 A5   D0   1F64  6131          MOVL    UCB$L_CRB(R5),R3         ; Get CRB address
                        1F68  6132          ASSUME  VEC$W_MAPREG+2 EQ VEC$B_NUMREG
                        1F68  6133          ASSUME  VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
         011C C4   D0   1F68  6134          MOVL    CDB_L_RINGMAP(R4),-      ; Setup map info in CRB
           34 A3        1F6C  6135           CRB$L_INTD+VEC$W_MAPREG(R3) ;
              0A   13   1F6E  6136          BEQL    50$                      ; Br if none
         011C C4   D4   1F70  6137          CLRL    CDB_L_RINGMAP(R4)        ; No more mapping info
                        1F74  6138          RELMPR                           ; Release the map register
                        1F7A  6139 50$:     ;
                        1F7A  6140          ; Deallocate all receive buffers and complete all I/O request packets
                        1F7A  6141          ;
      56   00DC C4   9E   1F7A  6142          MOVAB   CDB_Q_QUEUES(R4),R6      ; Get address of first queue listhead
           57   05   3C   1F7F  6143          MOVZWL  #CDB_C_QUEUES,R7         ; Get number of queues
        53   00 B6   0F   1F82  6144 60$:     REMQUE  @(R6)+,R3                ; Get next IRP/BUFFER
                  79   1D   1F86  6145          BVS     150$                     ; Br if none
        0A   0A A3   91   1F88  6146          CMPB    IRP$B_TYPE(R3),S^#DYN$C_IRP ; Is this an IRP?
                  0A   13   1F8C  6147          BEQL    70$                      ; Br if yes
        1B   0A A3   91   1F8E  6148          CMPB    IRP$B_TYPE(R3),S^#DYN$C_CXB ; Is this a CXB?
                  09   13   1F92  6149          BEQL    80$                      ; Br if yes
                        1F94  6150          BUG_CHECK NOBUFPCKT,FATAL        ; Else, fatal error
                        1F98  6151 ;
                        1F98  6152 ; IRP
                        1F98  6153 ;
               FB9E   30   1F98  6154 70$:     BSBW    ABORT_PKT                ; Abort the IRP
                 E5   11   1F9B  6155          BRB     60$                      ; Try for more
                        1F9D  6156 ;
                        1F9D  6157 ; CXB
                        1F9D  6158 ;
                        1F9D  6159 80$:     $DISPATCH       CXB$B_XQ_FUNC(R3),TYPE=B,-
                        1F9D  6160          <       ;function       action
                        1F9D  6161
                        1F9D  6162                  <XQ_FC_V_XMIT   100$>,- ; XMIT request
                        1F9D  6163                  <XQ_FC_V_RECV   140$>,- ; RECV request
                        1F9D  6164                  <XQ_FC_V_INIT   100$>,- ; INIT request
                        1F9D  6165                  <XQ_FC_V_STOP   100$>,- ; STOP request
                        1F9D  6166                  <XQ_FC_V_CHMODE 100$>,- ; Change mode request
                        1F9D  6167          >
                        1FB0  6168
                        1FB0  6169          BUG_CHECK NOBUFPCKT,FATAL        ; Fatal error - not a valid IRP
                        1FB4  6170
                        1FB4  6171 ;
                        1FB4  6172 ; CXB - XMIT request
                        1FB4  6173 ;
                        1FB4  6174 100$:    ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
          16 24 A3   E8   1FB4  6175          BLBS    CXB$L_T_IRP(R3),120$     ; Br if not IRP address => FFI user
      53   24 A3   D0   1FB8  6176          MOVL    CXB$L_T_IRP(R3),R3       ; Else, get IRP address
   50   0094 C3   D0   1FBC  6177          MOVL    IRP$L_XQ_SETUP(R3),R0    ; Get SETUP mode buffer
              06   13   1FC1  6178          BEQL    110$                     ; Br if not present
    00000000'GF   16   1FC3  6179          JSB     G^COM$DRVDEALMEM         ; Else, deallocate buffer
            FB6D   30   1FC9  6180 110$:    BSBW    ABORT_PKT                ; Abort the IRP
              B4   11   1FCC  6181          BRB     60$                      ; Try for more
                        1FCE  6182
                        1FCE  6183 120$:    ;
                        1FCE  6184          ;  This is an FFI user, return CXB buffer
                        1FCE  6185          ;
      51   18 A3   3C   1FCE  6186          MOVZWL  CXB$W_BOFF(R3),R1        ; Get offset to start of data
```

```
                51    53    C0  1FD2  6187                ADDL    R3,R1                           ; Compute start of Ethernet header
              51    0C A1    3C  1FD5  6188                MOVZWL  XBUF_W_TYPE(R1),R1              ; Get protocol type
                  0ABF      30  1FD9  6189                BSBW    MATCH_PROTYP                    ; Find the protocol user
                17 50       E9  1FDC  6190                BLBC    R0,140$                         ; Br if no UCB, drop buffer
                   54       DD  1FDF  6191                PUSHL   R4                              ; Save R4
            54    0192 C5    D0  1FE1  6192                MOVL    UCB$L_XQ_FFI(R5),R4             ; Get FFI block address
                      0B    13  1FE6  6193                BEQL    130$                            ; Br if not there, drop buffer
                50    2C    3C  1FE8  6194                MOVZWL  #SS$_ABORT,R0                   ; Set return status
                  14 B4     16  1FEB  6195                JSB     @FFI$L_XMIT_DONE(R4)            ; Complete the XMIT
                     54 8ED0  1FEE  6196                POPL    R4                              ; Restore R4
                     8F    11  1FF1  6197                BRB     60$                             ; Try for more
                               1FF3  6198 130$:           ;
                               1FF3  6199                ;  No more FFI for CXB to complete
                               1FF3  6200                ;
                     54 8ED0   1FF3  6201                POPL    R4                              ; Restore CDB address
                               1FF6  6202
                               1FF6  6203 ;
                               1FF6  6204 ; CXB - RECV request
                               1FF6  6205 ;
                               1FF6  6206
                50    53    D0  1FF6  6207 140$:   MOVL    R3,R0                           ; Copy CXB buffer address
            00000000'GF       16  1FF9  6208                JSB     G^COM$DRVDEALMEM                ; Deallocate the buffer
                     81    11  1FFF  6209 1060$:  BRB     60$                             ; Try for more
                               2001  6210                ;
                               2001  6211                ; Loop to next queue
                               2001  6212                ;
                56    08    C0  2001  6213 150$:   ADDL    #8,R6                           ; Skip to next queue listhead
                  F8 57       F5  2004  6214                SOBGTR  R7,1060$                        ; Loop if more queues
                               2007  6215                ;
                               2007  6216                ; Cleanup all I/O on all UNITS
                               2007  6217                ;
            55    0110 C4     D0  2007  6218                MOVL    CDB_L_UCB0(R4),R5               ; Get UNIT #0 UCB address
                      0B    13  200C  6219                BEQL    190$                            ; Br if none, yet
              55    30 A5     D0  200E  6220 170$:   MOVL    UCB$L_LINK(R5),R5               ; Get next unit's address
                      05    13  2012  6221                BEQL    190$                            ; Br if none
                    0026    30  2014  6222                BSBW    SHUTDOWN                        ; Shutdown the UNIT
                      F5    11  2017  6223                BRB     170$                            ; Check if more UNIT's
                 00E8 BF    BA  2019  6224 190$:   POPR    #^M<R3,R5,R6,R7>                ; Restore registers
                               201D  6225                RSB                                     ; Return to caller
```

```
                     201E  6227          .SBTTL SHUTDOWN - SHUT DOWN UNIT
                     201E  6228          .SBTTL SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
                     201E  6229 ;++
                     201E  6230 ; SHUTDOWN - SHUT DOWN UNIT
                     201E  6231 ; SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
                     201E  6232 ;
                     201E  6233 ; Functional description:
                     201E  6234 ;
                     201E  6235 ; This routine is used to shut down the XQ unit as a result of a
                     201E  6236 ; SETMODE and SHUTDOWN. The action is to abort all I/O for the unit
                     201E  6237 ; and then to clean up the unit data base.
                     201E  6238 ;
                     201E  6239 ; Inputs:
                     201E  6240 ;
                     201E  6241 ;       R3 = IRP address (SHUTDOWN_PROTYP entry only)
                     201E  6242 ;       R4 = CDB address
                     201E  6243 ;       R5 = UCB address
                     201E  6244 ;
                     201E  6245 ;       IPL = FIPL
                     201E  6246 ;
                     201E  6247 ; Outputs:
                     201E  6248 ;
                     201E  6249 ;       R3-R5 are preserved
                     201E  6250 ;
                     201E  6251 ;       R0-R2 are destroyed.
                     201E  6252 ;--
                     201E  6253 SHUTDOWN_PROTYP::                    ; Shut down protocol type
          04   E1    201E  6254          BBC     #UCB$V_ONLINE,-     ; Br if not online
 19 64 A5            2020  6255                  UCB$W_STS(R5),10$   ;
                     2023  6256          ASSUME  UCB$V_XQ_INITED  EQ  0
 15 68 A5   E9       2023  6257          BLBC    UCB$W_DEVSTS(R5),10$    ; Br if not inited
          03   E1    2027  6258          BBC     #UCB$V_XQ_SHARE,-       ; Br if not a shared UCB
 11 68 A5            2029  6259                  UCB$W_DEVSTS(R5),SHUTDOWN ; shutdown entire unit
                     202C  6260 ;
                     202C  6261 ; Try to find SHR data structure
                     202C  6262 ;
        E6E1   30    202C  6263          BSBW    MATCH_SHR               ; Check PID and CHAN
          0B   12    202F  6264          BNEQ    10$                     ; Br if NO MATCH, skip it
                     2031  6265 ;
                     2031  6266 ; Match found - clear inited bit and clean up all I/O on SHR data structure
                     2031  6267 ;
          56   DD    2031  6268          PUSHL   R6                      ; Save R6
    56    51   D0    2033  6269          MOVL    R1,R6                   ; Copy SHR address
        0252   30    2036  6270          BSBW    CLEANUP_SHR             ; Cleanup the SHR data structure
          56 8ED0    2039  6271          POPL    R6                      ; Restore R6
                05   203C  6272 10$:     RSB                             ; Return to caller
                     203D  6273
                     203D  6274 SHUTDOWN::                           ; Shut down unit
          04   E1    203D  6275          BBC     #UCB$V_ONLINE,-
 04 64 A5            203F  6276                  UCB$W_STS(R5),5$        ; If BC not online
                     2042  6277
                     2042  6278          ASSUME  UCB$V_XQ_INITED  EQ  0
 01 68 A5   E8       2042  6279          BLBS    UCB$W_DEVSTS(R5),10$    ; Br if UCB is inited
                05   2046  6280 5$:      RSB                             ; It's not time to shut down, yet
                     2047  6281
                     2047  6282 ;
                     2047  6283 ; If a power failure occurred, and the protocol has both initialized the FFI
```

```
                                2047  6284 ; interface and supplied an asynchronous error routine, then call back
                                2047  6285 ; the protocol at this routine address with a status value indicating that
                                2047  6286 ; a power failure had taken place.
                                2047  6287 ;
                                2047  6288
                     05   E1    2047  6289 10$:    BBC     #UCB$V_POWER,-          ; Skip notification if power failure
                19 64 A5         2049  6290                UCB$W_STS(R5),20$       ; did not occur
                     54   DD    204C  6291         PUSHL   R4                      ; Save CDB address
           54    0192 C5   D0    204E  6292         MOVL    UCB$L_XQ_FFI(R5),R4     ; Retrieve FFI address
                     0D   13    2053  6293         BEQL    15$                     ; Nothing to do if there isn't one
             52    1C A4   D0    2055  6294         MOVL    FFI$L_ERROR(R4),R2      ; Retrieve asynch error routine address
                     07   13    2059  6295         BEQL    15$                     ; Nothing to do if there isn't one
           50    0364 8F   3C    205B  6296         MOVZWL  #SS$_POWERFAIL,R0       ; Indicate that a powerfailure occurred
                     62   16    2060  6297         JSB     (R2)                    ; Call back the asynch error routine
                     54 8ED0    2062  6298 15$:    POPL    R4                      ; Restore CDB address
                                2065  6299
                                2065  6300 ;
                                2065  6301 ; Start a 3-second timer to restart any UNIT needing automatic restart.
                                2065  6302 ; This restart timer only runs if the device was halted due to a fatal error.
                                2065  6303 ;
                                2065  6304 ; Note, that the UCB multicast address list is purged, which will nullify
                                2065  6305 ; any restart operation that may be performed by the QNA driver itself (only
                                2065  6306 ; if the user has specified any multicast addresses).
                                2065  6307 ;
                                2065  6308
                00C8 8F    BB   2065  6309 20$:    PUSHR   #^M<R3,R6,R7>           ; Save registers
                     0F    E1   2069  6310         BBC     #UCB$V_XQ_RESTART,-     ; Br if this UNIT does not need
                43 68 A5         206B  6311                 UCB$W_DEVSTS(R5),22$    ;  automatic restart
                                206E  6312         ASSUME  IRP$C_LENGTH GE TQE$C_LENGTH
              51    F4 8F    9A   206E  6313         MOVZBL  #IRP$C_LENGTH+TQE$C_LENGTH,R1 ; Get size of IRP/TQE
          00000000'GF    16   2072  6314         JSB     G^EXE$ALONONPAGED       ; Try to allocate a IRP/TQE
                36 50    E9   2078  6315         BLBC    R0,22$                  ; Br if failure - too bad
                                207B  6316         ASSUME  IRP$Q_STATION GT TQE$C_LENGTH
                4000 8F    A8   207B  6317         BISW    #UCB$M_XQ_INTERLOCK,-   ; Interlock the RESTART bit
                  68 A5         207F  6318                 UCB$W_DEVSTS(R5)        ;
                                2081  6319
                                2081  6320         ASSUME  TQE$B_TYPE EQ TQE$W_SIZE+2
                                2081  6321         ASSUME  TQE$B_RQTYPE EQ TQE$B_TYPE+1
             000F00F4 8F    D0   2081  6322         MOVL    #<<DYN$C_TQE@16>!<IRP$C_LENGTH+TQE$C_LENGTH>>,- ; Set STRUCTURE
                  08 A2         2087  6323                 TQE$W_SIZE(R2)          ;  TYPE and SIZE
             00C8 C2   55   D0   2089  6324         MOVL    R5,TQE$C_LENGTH+IRP$L_RBOFF(R2) ; Save UCB address in IRP
                                208E  6325         PUSHQ   R4                      ; Save R4, R5
                55   52   D0   2091  6326         MOVL    R2,R5                   ; Copy TQE address
                          7D   2094  6327         MOVQ    #RESTART_DELTA,-        ; Set the delta time
          00 01C9C380 8F         2095
                     000000    209A
                  20 A5         209E  6328                 TQE$Q_DELTA(R5)         ;
             53   FD28 CF   9E   20A0  6329         MOVAB   W^RESTART_ROUT,R3       ; Get address of RESTART routine
                     01   90   20A5  6330         MOVB    #TQE$C_SSSNGL,-         ; Set the request type
                  2C A5         20A7  6331                 TQE$L_RQPID(R5)         ;
                   F275   30   20A9  6332         BSBW    FORK_TIMER              ; FORK to startup the timer
                                20AC  6333         POPQ    R4                      ; Restore R4, R5
                     05   11   20AF  6334         BRB     23$                     ; Contine
                                20B1  6335
                                20B1  6336 22$:    CLRBIT  #UCB$V_XQ_RESTART,-     ; Restart is not possible!
                                20B1  6337                  UCB$W_DEVSTS(R5)
                2328 8F    B0   20B6  6338 23$:    MOVW    #INIT_C_QUOTA,UCB$W_XQ_HBQ(R5) ; Reset hardware buffer quota
```

```
                00D6 C5          20BA
          64 A5    23    AA    20BD  6339          BICW    #UCB$M_INT!UCB$M_POWER!-
                                20C1  6340                  UCB$M_TIM,UCB$W_STS(R5) ; Reset device status
          68 A5    11    AA    20C1  6341          BICW    #UCB$M_XQ_INITED!-      ; No longer inited
                                20C5  6342                   UCB$M_XQ_RUN,UCB$W_DEVSTS(R5) ; ..or  running
                                20C5  6343          CLRBIT  #XM$V_STS_ACTIVE,UCB$L_DEVDEPEND(R5) ; Clear active bit
      010C C4    10    E1    20CA  6344          BBC     #XM$V_ERR_FATAL,CDB_L_DEVDEPEND(R4),25$ ; Br if not FATAL
                         05    20CF
                                20D0  6345          SETBIT  #XM$V_ERR_FATAL,UCB$L_DEVDEPEND(R5) ; Else, indicate FATAL
                                20D5  6346 ;
                                20D5  6347 ; Reset UCB multicast address list
                                20D5  6348 ;
      0210 C4    55    D1    20D5  6349 25$:    CMPL    R5,CDB_L_PRMUSER(R4)    ; Is this unit the PROMISCUOUS user?
                      0C    12    20DA  6350          BNEQ    27$                     ; Br if not
                0210 C4    D4    20DC  6351          CLRL    CDB_L_PRMUSER(R4)       ; Else, clear the PROMISCUOUS user addr
                      01    90    20E0  6352          MOVB    #NMA$C_STATE_OFF,-      ; Don't forget about the CDB
                0247 C4          20E2  6353                  CDB_B_PRM(R4)           ;  parameter
                   0154    30    20E5  6354          BSBW    BLD_STOP_IRP            ; Build an IRP to RESET hardware mode
                      0F    E0    20E8  6355 27$:     BBS     #UCB$V_XQ_RESTART,-     ; Br if this UNIT is restarting
             16 68 A5          20EA  6356                  UCB$W_DEVSTS(R5),28$    ;  don't clear multicast list
                                20ED  6357          PUSHQ   R4                      ; Save R4, R5
                00E9 C5    94    20F0  6358          CLRB    UCB$B_XQ_MULTI(R5)      ; No more multicast addresses
      00EC C5    00    2C    20F4  6359          MOVC5   #0,UCB$G_XQ_MULTI(R5),#0,- ; Zero the structure
                      00          20F9
                0048 8F          20FA  6360                  #6*MAX_C_MLT,UCB$G_XQ_MULTI(R5) ;
                00EC C5          20FD
                                2100  6361          POPQ    R4                      ; Restore R4, R5
                                2103  6362 ;
                                2103  6363 ; Reset CDB multicast address list, Flush all attention ASTs.
                                2103  6364 ;
                   038D    30    2103  6365 28$:     BSBW    ADD_MULTI               ; Re-calculate multicast address list
                      54    DD    2106  6366          PUSHL   R4                      ; Save CDB address
      57    00C0 C5    9E    2108  6367 30$:     MOVAB   UCB$L_XQ_AST(R5),R7     ; Get address of AST listhead
                50    67    D0    210D  6368          MOVL    (R7),R0                 ; Anything in list?
                      1B    13    2110  6369          BEQL    40$                     ; Br if not
               56    22 A0    3C    2112  6370          MOVZWL  ACB$L_KAST+10(R0),R6    ; Force channel match
               52    24 A0    3C    2116  6371          MOVZWL  ACB$L_KAST+12(R0),R2    ; Get process index
          00000000'GF    D0    211A  6372          MOVL    G^SCH$GL_PCBVEC,R4      ; Get PCB address vector address
                      54          2120
               54    6442    D0    2121  6373          MOVL    (R4)[R2],R4             ; Get PCB address
          00000000'GF    16    2125  6374          JSB     G^COM$FLUSHATTNS        ; Flush AST
                      DB    11    212B  6375          BRB     30$
                      54 8ED0    212D  6376 40$:     POPL    R4                      ; Restore CDB address
                                2130  6377 ;
                                2130  6378 ; Complete all RCV IRPs for this unit
                                2130  6379 ;
                                2130  6380          ASSUME  UCB$C_XQ_QUEUES-1 EQ 3  ; One queue for shared users
      53    00A8 D5    0F    2130  6381 45$:     REMQUE  @UCB$Q_XQ_RCVREQ(R5),R3 ; Get IRP
                      05    1D    2135  6382          BVS     50$                     ; Br if none
                   F9FF    30    2137  6383          BSBW    ABORT_PKT               ; Abort the I/O request
                      F4    11    213A  6384          BRB     45$                     ; Get next IRP
                                213C  6385 ;
                                213C  6386 ; Complete all XMIT CXBs for this unit
                                213C  6387 ;
      53    00B0 D5    0F    213C  6388 50$:     REMQUE  @UCB$Q_XQ_XMTREQ(R5),R3 ; Get CXB
                      2C    1D    2141  6389          BVS     55$                     ; Br if none
                                2143  6390          ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
```

```
                12 24 A3    E9  2143  6391          BLBC    CXB$L_T_IRP(R3),53$     ; Br if IRP address
                   54       DD  2147  6392          PUSHL   R4                      ; Save CDB address
54               0192 C5     D0  2149  6393          MOVL    UCB$L_XQ_FFI(R3),R4     ; Get FFI block address
                50   2C     3C  214E  6394          MOVZWL  #SS$_ABORT,R0           ; Set status return
                  14 B4      16  2151  6395          JSB     @FFI$L_XMIT_DONE(R4)    ; Complete CXB
                     54 8ED0  2154  6396          POPL    R4                      ; Restore CDB address
                     E3      11  2157  6397          BRB     50$                     ; Get next CXB
                             2159  6398
              53   24 A3     D0  2159  6399 53$:     MOVL    CXB$L_T_IRP(R3),R3      ; Get IRP address
50             0094 C3     D0  215D  6400          MOVL    IRP$L_XQ_SETUP(R3),R0   ; Get SETUP mode buffer
                     06      13  216C  6401          BEQL    54$                     ; Br if none
 00000000'GF         16      16  2164  6402          JSB     G^COM$DRVDEALMEM        ; Else, deallocate the buffer
                F9CC 30      216A  6403 54$:     BSBW    ABORT_PKT               ; Abort the I/O request
                     CD      11  216D  6404          BRB     50$                     ; Get next CXB
                             216F  6405 ;
                             216F  6406 ; Deallocate all receive CXBs
                             216F  6407 ;
52               00A0 D5     0F  216F  6408 55$:     REMQUE  @UCB$Q_XQ_RCVMSG(R5),R2 ; Get message buffer
                     24      1D  2174  6409          BVS     70$                     ; Br if none
                  42 A5      A0  2176  6410          ADDW    UCB$W_DEVBUFSIZ(R5),-   ; Restore quota
                00CC C5          2179  6411                  UCB$W_XQ_QUOTA(R5)      ;
                             217C  6412 ;
                             217C  6413 ;       The buffer may be smaller than the normal message size, if this
                             217C  6414 ;       is a cloned buffer for the promiscuous user. Therefore, we must
                             217C  6415 ;       check to make sure the buffer is large enough to be returned
                             217C  6416 ;       the the device's receive buffer pool.
                             217C  6417 ;
                     A1      217C  6418          ADDW3   #CXB$C_HEADER+-         ; Calculate size of "normal"
                             217D  6419                  CXB$C_TRAILER,-         ;  receive buffer
                004C 8F      217D  6420                  CDB_W_BSZ(R4),R0        ;
50              0108 C4      2180
  08 A2            50  B1  2184  6421          CMPW    R0,CXB$W_SIZE(R2)       ; Can buffer be returned?
                     05      12  2188  6422          BNEQ    60$                     ; Br if not, delete buffer instead
                   F2A8      30  218A  6423          BSBW    ADDRCVLIST              ; Try to add to receiver list
                     AD      11  218D  6424          BRB     50$                     ; Loop for more
                             218F  6425
              50     52      D0  218F  6426 60$:     MOVL    R2,R0                   ; Copy buffer address for deallocation
 00000000'GF         16      16  2192  6427          JSB     G^COM$DRVDEALMEM        ; Deallocate the buffer
                     A2      11  2198  6428          BRB     50$                     ; Loop for more
                             219A  6429 ;
                             219A  6430 ; Cleanup all SHR structures if fatal error
                             219A  6431 ;
                     03      E1  219A  6432 70$:     BBC     #UCB$V_XQ_SHARE,-       ; Br if not a SHARED UCB
                30 68 A5         219C  6433                  UCB$W_DEVSTS(R5),100$   ;
56              00C4 C5      D0  219F  6434          MOVL    UCB$L_XQ_DEFUSR(R5),R6  ; Get default SHR structure address
                     0B      13  21A4  6435          BEQL    90$                     ; Br if none
                   00E2      30  21A6  6436          BSBW    CLEANUP_SHR             ; Else, cleanup the structure
                     10      E1  21A9  6437          BBC     #XM$V_ERR_FATAL,-       ; Br if not a fatal error
                03 44 A5         21AB  6438                  UCB$L_DEVDEPEND(R5),90$
                   00FC      30  21AE  6439          BSBW    DELETE_SHR              ; And delete the structure
56              0098 C5      D0  21B1  6440 90$:     MOVL    UCB$Q_XQ_SHARE(R5),R6   ; Get address of next LIMITED user
0098 C5              56      D1  21B6  6441          CMPL    R6,UCB$Q_XQ_SHARE(R5)   ; End of list?
                     45      13  21BB  6442          BEQL    120$                    ; Br if yes, don't restore quota (yet)
                   00CB      30  21BD  6443          BSBW    CLEANUP_SHR             ; Cleanup the I/O
                     10      E1  21C0  6444          BBC     #XM$V_ERR_FATAL,-       ; Br if not a fatal error
                EC 44 A5         21C2  6445                  UCB$L_DEVDEPEND(R5),90$
                     0F      E0  21C5  6446          BBS     #UCB$V_XQ_RESTART,-     ; Br if this UNIT is re-starting
```

```
                                                I  2

                E7 68 A5          21C7  6447                UCB$W_DEVSTS(R5),90$
                    00E0  30      21CA  6448        BSBW    DELETE_SHR              ; Else, delete the structure
                      E2  11      21CD  6449        BRB     90$                     ; Look for more
                                  21CF  6450 ;
                                  21CF  6451 ; Restore quota
                                  21CF  6452 ;
                      0F  E0      21CF  6453 100$:  BBS     #UCB$V_XQ_RESTART,-     ; Br if this UNIT is re-starting
                5A 68 A5          21D1  6454                UCB$W_DEVSTS(R5),140$
           50   0088 C5    3C    21D4  6455        MOVZWL  UCB$L_XQ_PID(R5),R0     ; Get PID of last starter
            00000000'GF    D0    21D9  6456        MOVL    G^SCH$GL_PCBVEC,R1      ; Address PCB vector
                      51          21DF
              50   6140    D0    21E0  6457        MOVL    (R1)[R0],R0             ; Get PCB of owner
                   60 A0  D1    21E4  6458        CMPL    PCB$L_PID(R0),-
                0088 C5          21E7  6459                UCB$L_XQ_PID(R5)        ; Still there?
                      16  12    21EA  6460        BNEQ    120$                    ; If NEQ no
           50   0080 C0    D0    21EC  6461        MOVL    PCB$L_JIB(R0),R0        ; Get JIB address
           51   00CC C5    3C    21F1  6462        MOVZWL  UCB$W_XQ_QUOTA(R5),R1   ; Convert to longword
             20 A0    51  C0    21F6  6463        ADDL    R1,JIB$L_BYTCNT(R0)     ; Return byte count quota
             24 A0    51  C0    21FA  6464        ADDL    R1,JIB$L_BYTLM(R0)      ; ..and byte limit quota
                00CC C5    B4    21FE  6465        CLRW    UCB$W_XQ_QUOTA(R5)      ; Prevent this from being
                                  2202  6466                                        ; returned again
                                  2202  6467 ;
                                  2202  6468 ; Delete the STARTUP IRP for point-to-point mode
                                  2202  6469 ;
           50   0196 C5    D0    2202  6470 120$:  MOVL    UCD$L_XQ_STIRP(R5),R0   ;% Get the startup IRP address
                      0A  13    2207  6471        BEQL    130$                    ;% Br if none
                0196 C5    D4    2209  6472        CLRL    UCB$L_XQ_STIRP(R5)      ;% All done
            00000000'GF    16    220D  6473        JSB     G^COM$DRVDEALMEM        ;% Deallocate the IRP
                                  2213  6474 ;
                                  2213  6475 ; If there is an FFI block and the SHUT_DONE routine is set, then
                                  2213  6476 ; notify the FFI user that shutdown is now complete.
                                  2213  6477 ;
         50   0192 C5    D0    2213  6478 130$:  MOVL    UCB$L_XQ_FFI(R5),R0     ; Get FFI block address
                      14  13    2218  6479        BEQL    140$                    ; Br if none
                0192 C5    D4    221A  6480        CLRL    UCB$L_XQ_FFI(R5)        ; Cleanup FFI interface
           51     20 A0    D0    221E  6481        MOVL    FFI$L_SHUT_DONE(R0),R1  ; Get address of routine
                      0A  13    2222  6482        BEQL    140$                    ; Br if none
                      54  DD    2224  6483        PUSHL   R4                      ; Save CDB address
                54    50  D0    2226  6484        MOVL    R0,R4                   ; Copy FFI block address
                      61  16    2229  6485        JSB     (R1)                    ; Call back FFI user
                      54 8ED0   222B  6486        POPL    R4                      ; Restore CDB address
                                  222E  6487 ;
                                  222E  6488 ; Decrement UNIT count on CDB and cleanup CDB if last unit
                                  222E  6489 ;
                0208 C4    97    222E  6490 140$:  DECB    CDB_B_UNTCNT(R4)        ; One less unit on CDB
                      03  12    2232  6491        BNEQ    150$                    ; Br if more
                   FCD6   30    2234  6492        BSBW    SHUTDOWN_QNA            ; Else, shutdown entire QNA
                00C8 8F    BA    2237  6493 150$:  POPR    #^M<R3,R6,R7>           ; Restore registers
                      05  05    223B  6494        RSB                             ; Return to caller
                                  223C  6495
```

```
                     223C  6497          .SBTTL  BLD_STOP_IRP - Build an IRP to reset promiscuous mode
                     223C  6498 ;++
                     223C  6499 ; BLD_STOP_IRP - Build an IRP to reset the promiscuous mode
                     223C  6500 ;
                     223C  6501 ; Functional description:
                     223C  6502 ;
                     223C  6503 ; This routine will allocate and build an IRP to reset the hardware mode
                     223C  6504 ; from promiscous.
                     223C  6505 ;
                     223C  6506 ; Inputs:
                     223C  6507 ;       R4 = CDB address
                     223C  6508 ;       R5 = UCB address
                     223C  6509 ;
                     223C  6510 ; Outputs:
                     223C  6511 ;       R0,R1,R2,R3 are destroyed.
                     223C  6512 ;--
                     223C  6513          .ENABL  LSB
                     223C  6514 BLD_STOP_IRP:                                ; Build an IRP to reset hardware mode
                     223C  6515 ;
                     223C  6516 ; NOTE - we must use EXE$ALONONPAGED to allocate the IRP because the other
                     223C  6517 ;       routines reset the IPL to ASTDEL.
                     223C  6518 ;
          10   E0   223C  6519          BBS     #XM$V_ERR_FATAL,-       ; Br if fatal error,
 0E 010C C4         223E  6520                  CDB_L_DEVDEPEND(R4),10$ ;  ignore reset of mode
51   00C4 8F   3C   2242  6521          MOVZWL  #IRP$C_LENGTH,R1        ; Set length of IRP
 00000000'GF   16   2247  6522          JSB     G^EXE$ALONONPAGED       ; Try to allocate an IRP
       01 50   E8   224D  6523          BLBS    R0,20$                  ; Okay if buffer allocated
              05   2250  6524 10$:      RSB                             ; Else, too bad if we can't do it
                   2251  6525
  08 A2    51   B0   2251  6526 20$:     MOVW    R1,IRP$W_SIZE(R2)       ; Fill in the size field
              1E   10   2255  6527          BSBB    BLD_IRP                 ; Build a template IRP
0C A3   6C'AF   9E   2257  6528          MOVAB   B^DELETE_BLOCK,IRP$L_PID(R3) ; Store return address from IOPOST
         F105   30   225C  6529          BSBW    SETUP_MODE              ; Allocate setup mode buffer
       05 50   E9   225F  6530          BLBC    R0,70$                  ; Leave on error
          03   90   2262  6531          MOVB    #XQ_FC_V_STOP,-         ; Set function request,
       20 A2        2264  6532                  CXB$B_XQ_FUNC(R2)       ;  looks like a STOP
              05   2266  6533          RSB                             ; Return to queue request to DEQNA
                   2267  6534
    50    53   D0   2267  6535 70$:     MOVL    R3,R0                   ; Copy IRP address
          03   11   226A  6536          BRB     90$                     ; Deallocate IRP
                   226C  6537
                   226C  6538 DELETE_BLOCK:                            ; Deallocate a data structure
    50    55   D0   226C  6539          MOVL    R5,R0                   ; Get address of structure
00000000'GF   17   226F  6540 90$:     JMP     G^COM$DRVDEALMEM        ; Deallocate the structure
                   2275  6541
                   2275  6542          .DSABL  LSB
```

```
                        2275  6544          .SBTTL  BLD_STRT_IRP - Build a point-to-point startup IRP
                        2275  6545 ;++
                        2275  6546 ; BLD_STRT_IRP - Build a point-to-point startup IRP
                        2275  6547 ;
                        2275  6548 ; Functional description:
                        2275  6549 ;
                        2275  6550 ; This routine will build an IRP to perform a datalink startup with a
                        2275  6551 ; remote system.
                        2275  6552 ;
                        2275  6553 ; Inputs:
                        2275  6554 ;       R2 = IRP address
                        2275  6555 ;       R4 = CDB address
                        2275  6556 ;       R5 = UCB address
                        2275  6557 ;
                        2275  6558 ; Outputs:
                        2275  6559 ;       R0,R1,R2,R3 are destroyed.
                        2275  6560 ;--
;RNG0001                 2275    .2
```

```
                     2275  6588          .SBTTL  BLD_IRP - Build an IRP routine
                     2275  6589 ;++
                     2275  6590 ; BLD_IRP - Build an IRP routine
                     2275  6591 ;
                     2275  6592 ; Functional description:
                     2275  6593 ;
                     2275  6594 ; This routine will build a simple IRP and allow the caller to fill in the
                     2275  6595 ; function requested and then queue it to the DEQNA.
                     2275  6596 ;
                     2275  6597 ; Inputs:
                     2275  6598 ;       R2 = IRP address
                     2275  6599 ;       R5 = UCB address
                     2275  6600 ;
                     2275  6601 ; Outputs:
                     2275  6602 ;       R3 = IRP address
                     2275  6603 ;       R0-R2 are destroyed.
                     2275  6604 ;       R4,R5 are preserved.
                     2275  6605 ;--
                     2275  6606 BLD_IRP:                                   ; Build an IRP
53   82    7E        2275  6607          MOVAQ   (R2)+,R3                  ; Save IRP address, skip to size field
                     2278  6608          ASSUME  IRP$W_SIZE EQ 8
                     2278  6609          ASSUME  IRP$B_TYPE EQ IRP$W_SIZE+2
                     2278  6610          ASSUME  IRP$B_RMOD EQ IRP$B_TYPE+1
     82    B5        2278  6611          TSTW    (R2)+                     ; Skip SIZE
82   0A    B0        227A  6612          MOVW    #DYN$C_IRP,(R2)+          ; Make it look like an IRP
                     227D  6613          ASSUME  IRP$L_PID EQ IRP$B_RMOD+1
                     227D  6614          ASSUME  IRP$L_AST EQ IRP$L_PID+4
     82    7C        227D  6615          CLRQ    (R2)+                     ; Clear PID, AST
                     227F  6616          ASSUME  IRP$L_ASTPRM EQ IRP$L_AST+4
                     227F  6617          ASSUME  IRP$L_WIND EQ IRP$L_ASTPRM+4
     82    7C        227F  6618          CLRQ    (R2)+                     ; Clear ASTPRM, WIND
                     2281  6619          ASSUME  IRP$L_UCB EQ IRP$L_WIND+4
82   55    D0        2281  6620          MOVL    R5,(R2)+                  ; Store UCB address
                     2284  6621          ASSUME  IRP$W_FUNC EQ IRP$L_UCB+4
                     2284  6622          ASSUME  IRP$B_EFN EQ IRP$W_FUNC+2
                     2284  6623          ASSUME  IRP$B_PRI EQ IRP$B_EFN+1
                     2284  6624          ASSUME  IRP$L_IOSB EQ IRP$B_PRI+1
     82    7C        2284  6625          CLRQ    (R2)+                     ; Clear FUNC, EFN, PRI, IOSB
                     2286  6626          ASSUME  IRP$W_CHAN EQ IRP$L_IOSB+4
                     2286  6627          ASSUME  IRP$W_STS EQ IRP$W_CHAN+2
                     2286  6628          ASSUME  IRP$L_SVAPTE EQ IRP$W_STS+2
     82    7C        2286  6629          CLRQ    (R2)+                     ; Clear CHAN, STS, SVAPTE
                     2288  6630          ASSUME  IRP$W_BOFF EQ IRP$L_SVAPTE+4
                     2288  6631          ASSUME  IRP$W_BCNT EQ IRP$W_BOFF+2
                     2288  6632          ASSUME  IRP$L_BCNT EQ IRP$W_BCNT
     82    7C        2288  6633          CLRQ    (R2)+                     ; Clear BOFF, BCNT
           05        228A  6634          RSB                               ; Return to caller
                     228B  6635
```

```
                     228B  6637          .SBTTL  CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE
                     228B  6638          .SBTTL  DELETE_SHR - DELETE SHR DATA STRUCTURE
                     228B  6639 ;++
                     228B  6640 ; CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE
                     228B  6641 ;
                     228B  6642 ; This routine aborts all read request in progress and return all message
                     228B  6643 ; buffers back to the CDB structure for re-use.
                     228B  6644 ;
                     228B  6645 ; Inputs:
                     228B  6646 ;
                     228B  6647 ;       R4 = CDB address
                     228B  6648 ;       R5 = UCB address
                     228B  6649 ;       R6 = SHR address
                     228B  6650 ;
                     228B  6651 ;       IPL = FIPL
                     228B  6652 ;
                     228B  6653 ; Outputs:
                     228B  6654 ;
                     228B  6655 ;       R0-R2 are destroyed.
                     228B  6656 ;       All other registers are preserved.
                     228B  6657 ;--
                     228B  6658 CLEANUP_SHR::                            ; Cleanup all I/O on SHR structure
                  53 DD  228B  6659          PUSHL   R3                  ; Save R3
                     228D  6660
-1                   228D  6693
                     228D  6694          ASSUME  SHR_C_QUEUES EQ 2
                     228D  6695 ;
                     228D  6696 ; Complete all IRPs for this structure
                     228D  6697 ;
                  53 20 B6   0F  228D  6698 10$:    REMQUE  @SHR_Q_RCVREQ(R6),R3    ; Get IRP
                        05   1D  2291  6699          BVS     20$                 ; Br if none
                      F8A3   30  2293  6700          BSBW    ABORT_PKT           ; Abort the I/O request
                        F5   11  2296  6701          BRB     10$                 ; Get next IRP
                             2298  6702 ;
                             2298  6703 ; Deallocate all message blocks
                             2298  6704 ;
                  52 18 B6   0F  2298  6705 20$:    REMQUE  @SHR_Q_RCVMSG(R6),R2    ; Get message buffer
                        0B   1D  229C  6706          BVS     30$                 ; Br if none
                     42 A5   A0  229E  6707          ADDW    UCB$W_DEVBUFSIZ(R5),-   ; Restore quota
                   00CC C5       22A1  6708                   UCB$W_XQ_QUOTA(R5)      ;
                      F18E   30  22A4  6709          BSBW    ADDRCVLIST          ; Try to add to receiver list
                        EF   11  22A7  6710          BRB     20$                 ; Loop
                     53 8ED0     22A9  6711 30$:    POPL    R3                  ; Restore R3
                           05    22AC  6712          RSB                         ; Return to caller
                                 22AD  6713
                                 22AD  6714 ;++
                                 22AD  6715 ; DELETE_SHR - DELETE SHARE DATA STRUCTURE
                                 22AD  6716 ;
                                 22AD  6717 ; This routine deallocates the SHR data structure to system pool.
                                 22AD  6718 ;
                                 22AD  6719 ; Inputs:
                                 22AD  6720 ;
                                 22AD  6721 ;       R5 = UCB address
                                 22AD  6722 ;       R6 = SHR address
                                 22AD  6723 ;
                                 22AD  6724 ;       IPL = FIPL
                                 22AD  6725 ;
```

```
                        22AD  6726 ; Outputs:
                        22AD  6727 ;
                        22AD  6728 ;       R0-R1 are destroyed.
                        22AD  6729 ;       All other registers are preserved.
                        22AD  6730 ;--
                        22AD  6731 DELETE_SHR::                                 ; Delete SHR data structure
          5C A5   B7    22AD  6732          DECW    UCB$W_REFC(R5)              ; One less user of the unit
00C4 C5     56   D1    22B0  6733          CMPL    R6,UCB$L_XQ_DEFUSR(R5)      ; Is this the default user?
            1C   13    22B5  6734          BEQL    30$                         ; Br if yes
51    0098 C5   9E    22B7  6735          MOVAB   UCB$Q_XQ_SHARE(R5),R1       ; Get address of SHARE queue
      50   61   D0    22BC  6736          MOVL    (R1),R0                     ; Get address of next in queue
      51   50   D1    22BF  6737 10$:     CMPL    R0,R1                       ; Back to front of list?
            4D   13    22C2  6738          BEQL    90$                         ; Br if none found
      50   56   D1    22C4  6739          CMPL    R6,R0                       ; Is this the one?
            05   13    22C7  6740          BEQL    20$                         ; Br if yes
      50   60   D0    22C9  6741          MOVL    (R0),R0                     ; Else, get next in queue
            F1   11    22CC  6742          BRB     10$                         ; And try for match
      50   60   0F    22CE  6743 20$:     REMQUE  (R0),R0                     ; Remove structure from list
            04   11    22D1  6744          BRB     40$                         ; And delete the structure
      00C4 C5   D4    22D3  6745 30$:     CLRL    UCB$L_XQ_DEFUSR(R5)         ; No more default user
 50    0C A6   3C    22D7  6746 40$:     MOVZWL  SHR_L_PID(R6),R0            ; Get PID SHR structure
 00000000'GF   D0    22DB  6747          MOVL    G^SCH$GL_PCBVEC,R1          ; Address PCB vector
            51         22E1
   50   6140   D0    22E2  6748          MOVL    (R1)[R0],R0                 ; Get PCB of owner
        60 A0   D1    22E6  6749          CMPL    PCB$L_PID(R0),-
        0C A6         22E9  6750                  SHR_L_PID(R6)               ; Still there?
            1B   12    22EB  6751          BNEQ    60$                         ; If NEQ no
5C    0080 C0   D0    22ED  6752          MOVL    PCB$L_JIB(R0),R0            ; Get JIB address
  51    28 A6   3C    22F2  6753          MOVZWL  SHR_W_QUOTA(R6),R1          ; Convert to longword
  20 A0   51   C0    22F6  6754          ADDL    R1,JIB$L_BYTCNT(R0)         ; Return byte count quota
  24 A0   51   C0    22FA  6755          ADDL    R1,JIB$L_BYTLM(R0)          ; ..and byte limit quota
00CC C5   51   A2    22FE  6756          SUBW    R1,UCB$W_XQ_QUOTA(R5)       ; Decrease the current quota
0190 C5   51   A2    2303  6757          SUBW    R1,UCB$W_XQ_TOTQUO(R5)      ;  and the total quota
      50   56   D0    2308  6758 60$:     MOVL    R6,R0                       ; Copy SHR structure address
 00000000'GF   17    230B  6759          JMP     G^COM$DRVDEALMEM            ; Deallocate the structure
                     2311  6760
                     2311  6761 ;
                     2311  6762 ; Bug check on error
                     2311  6763 ;
                     2311  6764 90$:     BUG_CHECK NOBUFPCKT,FATAL
```

```
                     2315  6766          .SBTTL  CANCEL - CANCEL I/O ON UNIT
                     2315  6767 ;++
                     2315  6768 ; CANCEL - CANCEL I/O ON UNIT
                     2315  6769 ;
                     2315  6770 ; Functional description:
                     2315  6771 ;
                     2315  6772 ; This routine is used to cancel specific or all I/O pending on an XQ unit.
                     2315  6773 ;
                     2315  6774 ; Inputs:
                     2315  6775 ;
                     2315  6776 ;       R2 = Channel index number
                     2315  6777 ;       R4 = PCB address (or zero)
                     2315  6778 ;       R5 = UCB address
                     2315  6779 ;       R8 = Cancel reason code (CAN$C_DASSGN or CAN$C_CANCEL)
                     2315  6780 ;
                     2315  6781 ;
                     2315  6782 ;       IPL = FIPL
                     2315  6783 ;
                     2315  6784 ; Outputs:
                     2315  6785 ;
                     2315  6786 ;       R3-R5 are preserved.
                     2315  6787 ;       R0-R2 are destroyed.
                     2315  6788 ;
                     2315  6789 ;--
                     2315  6790
                     2315  6791 CANCEL::                                    ; Cancel I/O
          00D8 8F    BB   2315  6792          PUSHR   #^M<R3,R4,R6,R7>       ; Save registers
               03    E1   2319  6793          BBC     #UCB$V_XQ_SHARE,-      ; Br if not a shared UCB
       17 68 A5          231B  6794                   UCB$W_DEVSTS(R5),2$    ;  perform regular $CANCEL
                         231E  6795 ;
                         231E  6796 ; Try to find SHR data structure
                         231E  6797 ;
                     0136  30   231E  6798          BSBW    FIND_SHR               ; Check PID and CHAN
                       12  12   2321  6799          BNEQ    2$                     ; Br if NO MATCH, maybe last $DASSGN
                         2323  6800 ;
                         2323  6801 ;       Match found - clear inited bit and clean up all I/O on SHR data
                         2323  6802 ;       structure.
                         2323  6803 ;
                         2323  6804 ;       We will Delete the SHR structure if this is a $DASSGN function
                         2323  6805 ;       request. We will get this function when called from SYS$DASSGN
                         2323  6806 ;       system service and so we will have to delete the SHR structure
                         2323  6807 ;       and decrement the reference count. Note that the reference count
                         2323  6808 ;       can never reach zero. Therefore, SYS$DASSGN will decrement the
                         2323  6809 ;       reference count on exit and we will be called again. This time
                         2323  6810 ;       there will be no match on the PID/CHAN and so the UCB will be
                         2323  6811 ;       cleaned up and deleted.
                         2323  6812 ;
                   56 51 D0   2323  6813          MOVL    R1,R6                  ; Copy SHR address
;RNG0001      54  00C8 C5 D0   2326     .1          MOVL    UCB$L_XQ_CDB(R5),R4    ; Get CDB address
-2                 FF5D  30   232B  6816          BSBW    CLEANUP_SHR            ; Cleanup the SHR data structure
                         232E  6817
                         232E  6818          ASSUME  CAN$C_DASSGN EQ 1
                      58 D7   232E  6819          DECL    R8                     ; Deassign request?
                      33 12   2330  6820          BNEQ    10$                    ; Br if no - all done
                    FF78 30   2332  6821          BSBW    DELETE_SHR             ; Else, delete the SHR data stucture
                         2335  6822                                             ; And NOW perform like a NON-SHARED
                         2335  6823                                             ; unit.
```

```
                                2335  6824 ;
                                2335  6825 ; Non-shared unit - perform $CANCEL function.
                                2335  6826 ;
                     5C A5   B5 2335  6827 2$:     TSTW    UCB$W_REFC(R5)          ; Last reference?
                        30   12 2338  6828         BNEQ    20$                     ; Br if no - do selective cancel
;RNG0001          54   00C8 C5 DO 233A    .1 3$:     MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
;RNG0001                 FCFB 30 233F    .2         BSBW    SHUTDOWN                ; Shutdown entire unit
;RNG0001                         2342    .3         ASSUME  CDB_STS_V_INITED  EQ  0
;RNG0001             0A 0246 C4 E8 2342    .4         BLBS    CDB_B_STS(R4),5$        ; Br if QNA is still inited
-5                  024A C4    01 CE 2347  6834         MNEGL   #1,CDB_G_PHA(R4)        ; Reset physical address
                    024E C4    01 AE 234C  6835         MNEGW   #1,CDB_G_PHA+4(R4)      ;       ...
                                2351  6836 ;
                                2351  6837 ; When this is the last reference to the unit, reset the CPID of the UCB.
                                2351  6838 ;
                        10   A8 2351  6839 5$:     BISW    S^#UCB$M_ONLINE,-       ; Set the UNIT to ONLINE
                     64 A5      2353  6840                 UCB$W_STS(R5)           ;
                   00BC C5   D5 2355  6841         TSTL    UCB$L_XQ_CPID(R5)       ; Did we save the Creator PID?
                        0A   13 2359  6842         BEQL    10$                     ; Br if not
                   00BC C5   D0 235B  6843         MOVL    UCB$L_XQ_CPID(R5),UCB$L_CPID(R5) ; Else, restore Creator PID
                     20 A5      235F
                   00BC C5   D4 2361  6844         CLRL    UCB$L_XQ_CPID(R5)       ; Never again!!
                   00D8 8F   BA 2365  6845 10$:    POPR    #^M<R3,R4,R6,R7>        ; Restore registers
                        05   05 2369  6846         RSB
                                236A  6847 ;
                                236A  6848 ; Abort all associated receive packets on UCB queue
                                236A  6849 ;
;RNG0001            54    04 AE DO 236A    .1 20$:    MOVL    4(SP),R4                ; Restore PCB address to R4
;RNG0001                    04 E1 236E    .2         BBC     #UCB$V_ONLINE,-         ; Br if not online
-1                    F2 64 A5      2370  6851                  UCB$W_STS(R5),10$      ;
                                2373  6852
                                2373  6853         ASSUME  UCB$V_XQ_INITED  EQ  0
                      EE 68 A5 E9 2373  6854         BLBC    UCB$W_DEVSTS(R5),10$    ; Br if not inited
                   56   00A8 C5 9E 2377  6855         MOVAB   UCB$Q_XQ_RCVREQ(R5),R6  ; Get address of receive queue
                        29   10 237C  6856         BSBB    CHECKER                 ; Check packets on queue
                                237E  6857 ;
                                237E  6858 ; Abort all xmit requests on CDB queue
                                237E  6859 ;
;RNG0001            57   00C8 C5 DO 237E    .1         MOVL    UCB$L_XQ_CDB(R5),R7     ; Get CDB address
;RNG0001                         2383    .2         ASSUME  CDB_STS_V_INITED  EQ  0
;RNG0001               DD 0246 C7 E9 2383    .3         BLBC    CDB_B_STS(R7),10$       ; Br if not inited
;RNG0001                         2388    .4         DSBINT  UCB$B_DIPL(R5)          ; Sync access to CDB
;RNG0001            57   00DC C7 9E 238F    .5         MOVAB   CDB_Q_QUEUES(R7),R7     ; Get start of queues
;RNG0001                 58   01 9A 2394    .6         MOVZBL  #CDB_C_ABORTS,R8        ; Get number of queues we can abort on
-7                      56   57 DO 2397  6867 30$:    MOVL    R7,R6                   ; Set address of next queue
                             48 10 239A  6868         BSBB    CXB_CHECKER             ; Check CXBs on this queue
                        57   08 C0 239C  6869         ADDL    #8,R7                   ; Skip to next queue
                          F5 58 F5 239F  6870         SOBGTR  R8,30$                  ; Loop thru queues
                                23A2  6871         ENBINT                          ; Enable interrupts
                             BE   11 23A5  6872         BRB     10$                     ; Exit from cancel
                                23A7  6873 ;
                                23A7  6874 ; Subroutine to scan queue for match on all packets
                                23A7  6875 ;
                                23A7  6876 CHECKER:
                        53   66   D0 23A7  6877         MOVL    (R6),R3                 ; Get next entry
                        56   53   D1 23AA  6878 10$:    CMPL    R3,R6                   ; End of list?
                             11   13 23AD  6879         BEQL    30$                     ; Br if yes
                             10   10 23AF  6880         BSBB    CHECKPKT                ; Cancel if appropriate match
```

```
                     08   12   23B1  6881          BNEQ    20$                             ; Br if no match
                 53  63   0F   23B3  6882          REMQUE  (R3),R3                         ; Remove from list
                    F780   30   23B6  6883          BSBW    ABORT_PKT                       ; Complete the I/O request
                     EC   11   23B9  6884          BRB     CHECKER                         ; Look for more
                 53  63   D0   23BB  6885 20$:      MOVL    (R3),R3                         ; Travel link
                     EA   11   23BE  6886          BRB     10$                             ; Look for more
                          05   23C0  6887 30$:      RSB                                     ; Return to caller
                               23C1  6888 ;
                               23C1  6889 ; Subroutine to check for specific cancel
                               23C1  6890 ;
                               23C1  6891 CHECKPKT:
                   0C  A3   D5   23C1  6892          TSTL    IRP$L_PID(R3)                   ; Is this an Internal IRP?
                     11   19   23C4  6893          BLSS    30$                             ; Br if yes
                     06   12   23C6  6894          BNEQ    10$                             ; Br if valid PID
                               23C8  6895
                     54   D5   23C8  6896          TSTL    R4                              ; Valid PCB?
                     17   12   23CA  6897          BNEQ    50$                             ; Br if yes, no match
                     11   11   23CC  6898          BRB     40$                             ; Else, test CHAN
                               23CE  6899
              0C  A3   60  A4   D1   23CE  6900 10$:      CMPL    PCB$L_PID(R4),IRP$L_PID(R3) ; PID match?
                     0E   12   23D3  6901          BNEQ    50$                             ; Br if no
                     08   11   23D5  6902          BRB     40$                             ; Try CHAN match
                               23D7  6903
                   60  A4   D1   23D7  6904 30$:      CMPL    PCB$L_PID(R4),UCB$L_XQ_PID(R5) ; IS this the starter's PID?
                 00B8  C5        23DA
                     04   12   23DD  6905          BNEQ    50$                             ; Br if no
                28  A3   52   B1   23DF  6906 40$:      CMPW    R2,IRP$W_CHAN(R3)           ; Channel match?
                          05   23E3  6907 50$:      RSB                                     ; Return to caller
                               23E4  6908
                               23E4  6909 CXB_CHECKER:
                 53   66   D0   23E4  6910          MOVL    (R6),R3                         ; Get next entry
                 56   53   D1   23E7  6911 10$:      CMPL    R3,R6                           ; End of list?
                     38   13   23EA  6912          BEQL    30$                             ; Br if yes
                     37   10   23EC  6913          BSBB    CXB_CHECKPKT                    ; Cancel if appropriate match
                     2F   12   23EE  6914          BNEQ    20$                             ; Br if no match
                 53  63   0F   23F0  6915          REMQUE  (R3),R3                         ; Remove from CXB list
                               23F3  6916          ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
               16 24  A3   E8   23F3  6917          BLBS    CXB$L_T_IRP(R3),16$             ; Br if not IRP address -> FAST interface
              53  24  A3   D0   23F7  6918          MOVL    CXB$L_T_IRP(R3),R3              ; Else, get IRP address
            50  0094  C3   D0   23FB  6919          MOVL    IRP$L_XQ_SETUP(R3),R0           ; Get address of SETUP mode buffer
                     06   13   2400  6920          BEQL    13$                             ; Br if none
             00000000'GF   16   2402  6921          JSB     G^COM$DRVDEALMEM                ; Else, deallocate the buffer
                    F72E   30   2408  6922 13$:      BSBW    ABORT_PKT                       ; Complete the I/O request
                     D7   11   240B  6923          BRB     CXB_CHECKER                     ; Look for more
                     54   DD   240D  6924 16$:      PUSHL   R4                              ; Save R4
            54   0192  C5   D0   240F  6925          MOVL    UCB$L_XQ_FFI(R5),R4             ; Get FFI block address
                 50  2C   3C   2414  6926          MOVZWL  #SS$_ABORT,R0                   ; Set status return
                   14  B4   16   2417  6927          JSB     @FFI$L_XMIT_DONE(R4)            ; Complete the XMIT CXB
                     54  8ED0   241A  6928          POPL    R4                              ; Restore R4
                     C5   11   241D  6929          BRB     CXB_CHECKER                     ; Look for more
                 53  63   D0   241F  6930 20$:      MOVL    (R3),R3                         ; Travel link
                     C3   11   2422  6931          BRB     10$                             ; Look for more
                          05   2424  6932 30$:      RSB                                     ; Return to caller
                               2425  6933 ;
                               2425  6934 ; Subroutine to check for specific cancel
                               2425  6935 ;
                               2425  6936 CXB_CHECKPKT:
```

```
                50   24 A3    D0  2425  6937          MOVL    CXB$L_T_IRP(R3),R0      ; Get (presumed) IRP address
                                  2429  6938          ASSUME  CXB$L_T_IRP  EQ  CXB$L_T_UCB
                     23 50    E8  2429  6939          BLBS    R0,80$                  ; Br if not an IRP address
                     0C A0    D5  242C  6940          TSTL    IRP$L_PID(R0)           ; Is this an Internal IRP?
                         11   19  242F  6941          BLSS    30$                     ; Br if yes
                         06   12  2431  6942          BNEQ    10$                     ; Br if valid PID
                                  2433  6943
                         54   D5  2433  6944          TSTL    R4                      ; Valid PCB?
                         17   12  2435  6945          BNEQ    50$                     ; Br if yes, no match
                         11   11  2437  6946          BRB     40$                     ; Else, test CHAN
                                  2439  6947
             0C A0       60 A4    D1  2439  6948 10$:  CMPL    PCB$L_PID(R4),IRP$L_PID(R0) ; PID match?
                         0E   12  243E  6949          BNEQ    50$                     ; Br if no
                         08   11  2440  6950          BRB     40$                     ; Try CHAN match
                                  2442  6951
                      60 A4    D1  2442  6952 30$:  CMPL    PCB$L_PID(R4),UCB$L_XQ_PID(R5) ; IS this the starter's PID?
                   00B8 C5        2445
                         04   12  2448  6953          BNEQ    50$                     ; Br if no
                28 A0    52   B1  244A  6954 40$:  CMPW    R2,IRP$W_CHAN(R0)       ; Channel match?
                         05   05  244E  6955 50$:  RSB                             ; Return to caller
                                  244F  6956
                                  244F  6957          ; No IRP with CXB - FFI user
                                  244F  6958
                         54   D5  244F  6959 80$:  TSTL    R4                      ; No PCB?
                         FB   13  2451  6960          BEQL    50$                     ; Br if true - abort I/O
                   53    53   D0  2453  6961          MOVL    R3,R3                   ; Else, return Z-BIT clear
                         05   05  2456  6962          RSB                             ; Return to caller
                                  2457  6963
```

```
                        2457  6965          .SBTTL  SUBROUTINES TO FIND SHR DATA STRUCTURE GIVEN PCB AND CHAN
                        2457  6966 ;+
                        2457  6967 ; Subroutine to find SHR data structure for user
                        2457  6968 ;
                        2457  6969 ; Inputs:
                        2457  6970 ;       R2 = Channel number
                        2457  6971 ;       R4 = PCB address (or zero)
                        2457  6972 ;       R5 = UCB address
                        2457  6973 ;
                        2457  6974 ; Outputs:
                        2457  6975 ;       R1 = Address if SHR data structure if match
                        2457  6976 ;       R0 is destroyed.
                        2457  6977 ;       Z-Bit set then match.
                        2457  6978 ;       Z-Bit clear then no match.
                        2457  6979 ;-
                        2457  6980
                        2457  6981 FIND_SHR:                                    ; Try to find shared user
51   00C4 C5    D0      2457  6982          MOVL    UCB$L_XQ_DEFUSR(R5),R1     ; Get address of default user
           04    13      245C  6983          BEQL    10$                        ; Br if no default user
           1C    10      245E  6984          BSBB    90$                        ; Check for match
           19    13      2460  6985          BEQL    40$                        ; Br if match
50   0098 C5    9E      2462  6986 10$:      MOVAB   UCB$Q_XQ_SHARE(R5),R0      ; Save address of listhead
     51    50    D0      2467  6987          MOVL    R0,R1                      ; Copy listhead address
                        246A  6988          ASSUME  SHR_L_QFL EQ 0
     51    61    D0      246A  6989 20$:      MOVL    (R1),R1                    ; Get next in list
     50    51    D1      246D  6990          CMPL    R1,R0                      ; Back to start of list?
           06    13      2470  6991          BEQL    30$                        ; Br if yes - no pid/chan match
           08    10      2472  6992          BSBB    90$                        ; Check for match
           F4    12      2474  6993          BNEQ    20$                        ; Br if none
           03    11      2476  6994          BRB     40$                        ; Return in success
     50    50    D0      2478  6995 30$:      MOVL    R0,R0                      ; Return match failure
           05    247B  6996 40$:      RSB
                 247C  6997
                 247C  6998 ;+
                 247C  6999 ; Subroutine to check if PID and SHR data base match up
                 247C  7000 ;
                 247C  7001 ; Inputs:
                 247C  7002 ;       R1 = SHR address
                 247C  7003 ;       R2 = Channel number
                 247C  7004 ;       R4 = PCB address (or zero)
                 247C  7005 ;
                 247C  7006 ; Outputs:
                 247C  7007 ;       Z-Bit set then match.
                 247C  7008 ;       Z-Bit clear then no match.
                 247C  7009 ;-
                 247C  7010
                 247C  7011 90$:                                            ; Check for match with SHR data base
     54    D5      247C  7012          TSTL    R4                             ; Valid PCB address?
           07    12      247E  7013          BNEQ    100$                       ; Br if yes
     0C A1 D5      2480  7014          TSTL    SHR_L_PID(R1)                  ; Zero PID?
           0D    12      2483  7015          BNEQ    140$                       ; Br if not
           07    11      2485  7016          BRB     110$                       ; Try for CHAN
0C A1      60 A4 D1      2487  7017 100$:     CMPL    PCB$L_PID(R4),SHR_L_PID(R1) ; PIDs match?
           04    12      248C  7018          BNEQ    140$                       ; Br if no - try for next
   1C A1   52    B1      248E  7019 110$:     CMPW    R2,SHR_W_CHAN(R1)          ; Channels match?
           05    2492  7020 140$:     RSB                                    ; Return to caller
```

```
                        2493  7022          .SBTTL  FIND_POINT_UCB  - Find the point to point UCB
                        2493  7023 ;++
                        2493  7024 ; FIND_POINT_UCB - Find the point-to-point UCB
                        2493  7025 ;
                        2493  7026 ; Functional description:
                        2493  7027 ;
                        2493  7028 ; This routine is called to find the point-to-point UCB for some received
                        2493  7029 ; message. This is only needed when the protocol is in the startup state.
                        2493  7030 ;
                        2493  7031 ; Inputs:
                        2493  7032 ;
                        2493  7033 ;       R1 = Protocol type (startup)
                        2493  7034 ;       R2 = MSG buffer address
                        2493  7035 ;       R4 = CDB address
                        2493  7036 ;
                        2493  7037 ;
                        2493  7038 ;       IPL = FIPL
                        2493  7039 ;
                        2493  7040 ; Outputs:
                        2493  7041 ;
                        2493  7042 ;       R0 = Status return for request
                        2493  7043 ;       All other registers are preserved
                        2493  7044 ;--
:RNG0001                2493    .2
```

```
                                2493  7111          .SBTTL  ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
                                2493  7112 ;++
                                2493  7113 ; ADD_MULTI - ADD UP ALL THE MULTICAST ADDRESSES
                                2493  7114 ;
                                2493  7115 ; Functional description:
                                2493  7116 ;
                                2493  7117 ; This routine is called to combine all the per protocol type multicast
                                2493  7118 ; addresses into a single list in the CDB. If the sum of all multicast
                                2493  7119 ; addresses is greater than the QNA can manage, then an error is returned.
                                2493  7120 ;
                                2493  7121 ; Inputs:
                                2493  7122 ;
                                2493  7123 ;       R5 = UCB address
                                2493  7124 ;
                                2493  7125 ;
                                2493  7126 ;       IPL = FIPL
                                2493  7127 ;
                                2493  7128 ; Outputs:
                                2493  7129 ;
                                2493  7130 ;       R0 = Status return for request
                                2493  7131 ;       R1,R2 are destroyed
                                2493  7132 ;       R3-R5 are preserved
                                2493  7133 ;
                                2493  7134 ; Implicit outputs:
                                2493  7135 ;
                                2493  7136 ;       CDB_B_MLTTBL = Number of multicast addresses in CDB_G_MLTTBL
                                2493  7137 ;       CDB_G_MLTTBL = New multicast address list
                                2493  7138 ;--
                                2493  7139
                                2493  7140 ADD_MULTI:                                    ; Add up all the multicast addresses
                    00D8 8F     BB  2493  7141          PUSHR   #^M<R3,R4,R6,R7>        ; Save registers
;RNG0001            54   00C8 C5    D0  2497    .1          MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2                      025D C4     94  249C  7144          CLRB    CDB_B_MLTTBL(R4)        ; Reset number of entries
                                24A0  7145          PUSHQ   R4                      ; Save CDB and UCB addresses
                    02A6 C4     00  2C  24A3  7146          MOVC5   #0,CDB_G_MLTTBL(R4),#0,- ; Zero the structure
                                00      24A8
                        0048 8F         24A9  7147                  #6*MAX_C_MLT,CDB_G_MLTTBL(R4) ;
                        02A6 C4         24AC
                                24AF  7148          POPQ    R4                      ; Restore CDB and UCB addresses
                        53   0D     9A  24B2  7149          MOVZBL  #MAX_C_MLT+1,R3         ; Error if 1 more multicast address
                                24B5  7150                                          ;  than we can handle
                        50   01     CE  24B5  7151          MNEGL   #1,R0                   ; Assume success
                    56  02A6 C4     9E  24B8  7152          MOVAB   CDB_G_MLTTBL(R4),R6     ; Get address of Multicast table
                      57   28 A5     D0  24BD  7153          MOVL    UCB$L_DDB(R5),R7        ; Get DDB address
                      57   04 A7     D0  24C1  7154          MOVL    DDB$L_UCB(R7),R7        ; Get 1st UCB address
                      57   30 A7     D0  24C5  7155 10$:     MOVL    UCB$L_LINK(R7) R7       ; Get next UCB in list
                           37     13  24C9  7156          BEQL    50$                     ; Br if no more UCB's
                                24CB  7157
                                24CB  7158          ASSUME  UCB$V_XQ_INITED  EQ  0
                        F6 68 A5    E9  24CB  7159          BLBC    UCB$W_DEVSTS(R5),10$    ; Br if not inited
                    52  00EC C7    9E  24CF  7160          MOVAB   UCB$G_XQ_MULTI(R7),R2   ; Get address of Multicast list
                    51  00E9 C7    9A  24D4  7161          MOVZBL  UCB$B_XQ_MULTI(R7),R1   ; Set number addresses for UCB
                           62     D5  24D9  7162 20$:     TSTL    (R2)                    ; Is this field unused?
                           05     12  24DB  7163          BNEQ    25$                     ; Br if no
                             04 A2    B5  24DD  7164          TSTW    4(R2)                   ; Really?
                           16     13  24E0  7165          BEQL    30$                     ; Yes - skip it
                           53     97  24E2  7166 25$:     DECB    R3                      ; One less available slot in CDB
```

```
                1A  13  24E4  7167          BEQL    40$                     ; Br if none left - error
             86 62  D0  24E6  7168          MOVL    (R2),(R6)+              ; Else, insert next address
          86 04 A2  B0  24E9  7169          MOVW    4(R2),(R6)+             ;       ...
          025D C4   96  24ED  7170          INCB    CDB_B_MLTTBL(R4)        ; Count one more in list
          025D C4   91  24F1  7171          CMPB    CDB_B_MLTTBL(R4),-      ; Is there enough room?
                    0C  24F5  7172           #MAX_C_MLT                     ;       ...
                    08  1A  24F6  7173      BGTRU   40$                     ; Br if no - error
             52     06  C0  24F8  7174 30$:  ADDL    #6,R2                   ; Skip to next entry
               DB   51  F5  24FB  7175      SOBGTR  R1,20$                  ; Br if more
                    C5  11  24FE  7176      BRB     10$                     ; Else, skip to next UCB
                        2500  7177
                    50  D4  2500  7178 40$:  CLRL    R0                      ; Return failure
          00D8      8F  BA  2502  7179 50$:  POPR    #^M<R3,R4,R6,R7>        ; Restore registers
                        05  2506  7180      RSB
```

```
                     2507  7182          .SBTTL  MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
                     2507  7183 ;++
                     2507  7184 ; MOVE_MULTI - COPY THE MULTICAST ADDRESS LIST
                     2507  7185 ;
                     2507  7186 ; Functional description:
                     2507  7187 ;
                     2507  7188 ; This routine is called to copy the multicast address list from the
                     2507  7189 ; generation table to the actual list.
                     2507  7190 ;
                     2507  7191 ; Inputs:
                     2507  7192 ;
                     2507  7193 ;       R4 = CDB address
                     2507  7194 ;
                     2507  7195 ; Outputs:
                     2507  7196 ;
                     2507  7197 ;       All registers are preserved.
                     2507  7198 ;
                     2507  7199 ; Implicit outputs:
                     2507  7200 ;
                     2507  7201 ;       CDB_B_MULTI = Number of multicast addresses in CDB_G_MULTI
                     2507  7202 ;       CDB_G_MULTI = New multicast address list
                     2507  7203 ;--
                     2507  7204
                     2507  7205 MOVE_MULTI:                                  ; Move the multicast address list
        3F   BB      2507  7206          PUSHR   #^M<R0,R1,R2,R3,R4,R5>  ; Save registers
0048 8F      28      2509  7207          MOVC3   #MAX_C_MLT*6,CDB_G_MLTTBL(R4),CDB_G_MULTI(R4) ; Copy list
02A6 C4              250D
025E C4              2510
        3F   BA      2513  7208          POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore registers
025D C4      90      2515  7209          MOVB    CDB_B_MLTTBL(R4),CDB_B_MULTI(R4) ; Set number of entries
025C C4              2519
             05      251C  7210          RSB
```

```
                        251D  7212          .SBTTL  ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
                        251D  7213 ;++
                        251D  7214 ; ROUTINES TO SAVE/RESTORE UCB'S MULTICAST ADDRESS LIST
                        251D  7215 ;
                        251D  7216 ; Functional description:
                        251D  7217 ;
                        251D  7218 ; These routines are called to save or restore the multicast address list
                        251D  7219 ; in the UCB.
                        251D  7220 ;
                        251D  7221 ; Inputs:
                        251D  7222 ;
                        251D  7223 ;       R5 = UCB address
                        251D  7224 ;
                        251D  7225 ; Outputs:
                        251D  7226 ;
                        251D  7227 ;       All registers are preserved.
                        251D  7228 ;--
                        251D  7229
                        251D  7230 SAV_MULTI:                                   ; Save multicast address list
     3F   BB            251D  7231          PUSHR   #^M<R0,R1,R2,R3,R4,R5>     ; Save registers
0048 8F   28            251F  7232          MOVC3   #6*MAX_C_MLT,UCB$G_XQ_MULTI(R5),- ; Save multicast addresses
00EC C5                 2523
0134 C5                 2526  7233                  UCB$G_XQ_MLTTBL(R5)       ;       ...
     3F   BA            2529  7234          POPR    #^M<R0,R1,R2,R3,R4,R5>     ; Restore registers
00E9 C5   90            252B  7235          MOVB    UCB$B_XQ_MULTI(R5),-       ; Save count of multicast addresses
00EA C5                 252F  7236                  UCB$B_XQ_MLTTBL(R5)       ;       ...
          05            2532  7237          RSB                               ; Return to caller
                        2533  7238
                        2533  7239 RES_MULTI:                                   ; Restore multicast address list
     3F   BB            2533  7240          PUSHR   #^M<R0,R1,R2,R3,R4,R5>     ; Save registers
0048 8F   28            2535  7241          MOVC3   #6*MAX_C_MLT,UCB$G_XQ_MLTTBL(R5),- ; Restore multicast addresses
0134 C5                 2539
00EC C5                 253C  7242                  UCB$G_XQ_MULTI(R5)        ;       ...
     3F   BA            253F  7243  POPR    #^M<R0,R1,R2,R3,R4,R5>             ; Restore registers
00EA C5   90            2541  7244  MOVB    UCB$B_XQ_MLTTBL(R5),-              ; Restore count of multicast addresses
00E9 C5                 2545  7245                  UCB$B_XQ_MULTI(R5)        ;       ...
          05            2548  7246  RSB                                       ; Return to caller
```

```
                     2549  7248          .SBTTL  VALIDATE_P2 - VALIDATE P2 BUFFER PARAMETERS
                     2549  7249
                     2549  7250 ;++
                     2549  7251 ; VALIDATE_P2 - Validate P2 buffer parameters
                     2549  7252 ;
                     2549  7253 ; This routine is called to validate the P2 buffer parameters. The parameters
                     2549  7254 ; are checked against a parameter table which verifies that the minimum value
                     2549  7255 ; and maximum value is not violated, and that invalid status flags are not set.
                     2549  7256 ; If the parameter is a string, then the string must not exceed the maximum
                     2549  7257 ; string count for this parameter.
                     2549  7258 ;
                     2549  7259 ; Inputs:
                     2549  7260 ;
                     2549  7261 ;       R2 = Address of verification table
                     2549  7262 ;       R3 = IRP address
                     2549  7263 ;       R5 = UCB address
                     2549  7264 ;
                     2549  7265 ;
                     2549  7266 ; Outputs:
                     2549  7267 ;
                     2549  7268 ;       R0 = Status return for request
                     2549  7269 ;
                     2549  7270 ; If no error:
                     2549  7271 ;       R1 = Address of parameter verification table
                     2549  7272 ; If error:
                     2549  7273 ;       R1 = Bad parameter value
                     2549  7274 ;
                     2549  7275 ;       All other registers are preserved.
                     2549  7276 ;
                     2549  7277 ;--
                     2549  7278
                     2549  7279 VALIDATE_P2::                                ; Validate P2 buffer parameters
          03DC 8F    BB  2549  7280          PUSHR   #^M<R2,R3,R4,R6,R7,R8,R9> ; Save registers
     56    2C A3    D0  254D  7281          MOVL    IRP$L_SVAPTE(R3),R6       ; Get system P2 buffer address
           03   12  2551  7282          BNEQ    20$                       ; Br if a system buffer
         0121   31  2553  7283 10$:      BRW     150$                      ; Else, leave
                     2556  7284
;RNG0001   54  00C8 C5    D0  2556     .1 20$:     MOVL    UCB$L_XQ_CDB(R5),R4       ; Get CDB address
-2          56    66    D0  255B  7287          MOVL    P2B_L_POINTER(R6),R6      ; Point to start of P2 data
            58    32 A3    3C  255E  7288          MOVZWL  IRP$W_BCNT(R3),R8         ; Get size of P2 buffer
                     2562  7289 ;
                     2562  7290 ; Loop to check next parameter in P2 buffer
                     2562  7291 ;
            58    02    C2  2562  7292 30$:     SUBL    #2,R8                     ; Can we get another parameter?
                  EC   19  2565  7293          BLSS    10$                       ; Br if no - all done
            51    86    3C  2567  7294          MOVZWL  (R6)+,R1                  ; Get parameter type from P2
                     256A  7295 ;
                     256A  7296 ; *** NOTE - R2 MUST be the very first item on the top of the stack
                     256A  7297 ;
            57    6E    D0  256A  7298          MOVL    (SP),R7                   ; Get verification table address
                     256D  7299 ;
                     256D  7300 ; Loop to check P2 buffer parameter to Line parameter table
                     256D  7301 ;
                     256D  7302          ASSUME  PRM_W_TYPE EQ 0
            50    87    B0  256D  7303 40$:     MOVW    (R7)+,R0                  ; Get parameter type code
                  03   12  2570  7304          BNEQ    45$                       ; Br if NOT end of verify table
                0107   31  2572  7305 43$:     BRW     170$                      ; Else exit in error
```

```
                        2575  7306
                        2575  7307 45$:     ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
      59   87   9A       2575  7308          MOVZBL  (R7)+,R9                  ; Get flags byte
50    F000 8F   AA       2578  7309          BICW    #^C<PRM_TYP_M_CODE>,R0   ; Clear all but type code
      50   51   B1       257D  7310          CMPW    R1,R0                    ; Parameters match?
           17   13       2580  7311          BEQL    50$                      ; Br if yes
      57   02   C0       2582  7312          ADDL    #2,R7                    ; Skip offset word
                        2585  7313          SKIP    PRM_FLG_V_MIN,R9,R7      ; Skip minimum value
                        2588  7314          SKIP    PRM_FLG_V_MAX,R9,R7      ; Skip maximum value
                        2591  7315          SKIP    PRM_FLG_V_INVALID,R9,R7  ; Skip invalid flags
           D4   11      2597  7316          BRB     40$                      ; Try next parameter
                        2599  7317 ;
                        2599  7318 ; Match found - nullify if same value & check min,max,valid,invalid
                        2599  7319 ;
      50   87   B0       2599  7320 50$:     MOVW    (R7)+,R0                 ; Get offset + width
           0A   EF       259C  7321          EXTZV   #PRM_OFF_V_WIDTH,-       ; Get width only
52    50   06            259E  7322                  #PRM_OFF_S_WIDTH,R0,R2   ;   ...
           00   EF       25A1  7323          EXTZV   #PRM_OFF_V_VALUE,-       ; Get offset only
50    50   0A            25A3  7324                  #PRM_OFF_S_VALUE,R0,R0   ;   ...
  05  59   03   E0       25A6  7325          BBS     #PRM_FLG_V_CDB,R9,55$    ; Br if CDB datum
      50   55   C0       25AA  7326          ADDL    R5,R0                    ; Compute offset in UCB
           03   11       25AD  7327          BRB     57$                      ; Continue
      50   54   C0       25AF  7328 55$:     ADDL    R4,R0                    ; Compute offset in CDB
                        25B2  7329 57$:     ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
31 FB A7   0C   E0       25B2  7330          BBS     #PRM_TYP_V_STRING,-5(R7),95$ ; Br if string parameter
      58   04   C2       25B7  7331          SUBL    #4,R8                    ; Must be longword value
           B6   19       25BA  7332          BLSS    43$                      ; Br if error
      53   86   D0       25BC  7333          MOVL    (R6)+,R3                 ; Get parameter value
                        25BF  7334          CASE    R2,TYPE=B,LIMIT=#1,<-    ; Br to handler
                        25BF  7335                  60$,-                    ;   Byte value
                        25BF  7336                  70$,-                    ;   Word value
                        25BF  7337                  80$>                     ;   Longword value
                        25C9  7338 ;
                        25C9  7339 ; Byte value in structure
                        25C9  7340 ;
           60   53   91    25C9  7341 60$:     CMPB    R3,(R0)                  ; Is this the same?
                08   11    25CC  7342          BRB     90$                      ; Check result
                        25CE  7343 ;
                        25CE  7344 ; Word value
                        25CE  7345 ;
           60   53   B1    25CE  7346 70$:     CMPW    R3,(R0)                  ; Is this the same?
                03   11    25D1  7347          BRB     90$                      ; Check result
                        25D3  7348 ;
                        25D3  7349 ; Longword value
                        25D3  7350 ;
           60   53   D1    25D3  7351 80$:     CMPL    R3,(R0)                  ; Is this the same?
                6E   12    25D6  7352 90$:     BNEQ    100$                     ; Br if no - continue checks
0B0E       8F   51   B1    25D8  7353          CMPW    R1,#NMA$C_PCLI_PTY       ; Is this the protocol type?
                03   13    25DD  7354          BEQL    91$                      ; Br if yes - always store this
           FA   A6   B4    25DF  7355          CLRW    -6(R6)                   ; Nullify the parameter code
                008F 31    25E2  7356 91$:     BRW     140$                     ; Try next parameter - skip checks
                0094 31    25E5  7357 93$:     BRW     170$                     ; LONNGGG Branch to 170$
                        25E8  7358 ;
                        25E8  7359 ; String value
                        25E8  7360 ;
      58   02   C2       25E8  7361 95$:     SUBL    #2,R8                    ; Can we fetch string length?
           F8   19       25EB  7362          BLSS    93$                      ; Br if no - error
```

```
                53   86   3C  25ED  7363          MOVZWL  (R6)+,R3                 ; Get string length
                58   53   C2  25F0  7364          SUBL    R3,R8                   ; Is there room for string?
                     F0   19  25F3  7365          BLSS    93$                     ; Br if no - error
                52   53   B1  25F5  7366          CMPW    R3,R2                   ; Is the string too long?
                     EB   1A  25F8  7367          BGTRU   93$                     ; Br if yes - error
                56   53   C0  25FA  7368          ADDL    R3,R6                   ; Skip past string
           51   0B21 8F   B1  25FD  7369          CMPW    #NMA$C_PCLI_DES,R1      ; Is this the destination address?
                     07   13  2602  7370          BEQL    96$                     ; Br if yes
           51   0B04 8F   B1  2604  7371          CMPW    #NMA$C_PCLI_PHA,R1      ; Is this the physical address?
                     08   12  2609  7372          BNEQ    97$                     ; Br if not
                  026A    30  260B  7373  96$:    BSBW    VALID_PHYAD             ; Validate the physical address
                6B 50     E9  260E  7374          BLBC    R0,170$                 ; Br if error in physical address
                     61   11  2611  7375          BRB     140$                    ; Else, continue checking
           51   0B0F 8F   B1  2613  7376  97$:    CMPW    #NMA$C_PCLI_MCA,R1      ; Is this the multicast address list?
                     3E   12  2618  7377          BNEQ    130$                    ; Br if no - okay
                  021F    30  261A  7378          BSBW    VALID_MULTI             ; Validate the multicast address list
                5C 50     E9  261D  7379          BLBC    R0,170$                 ; Br if error
                          2620  7380          DSBINT  UCB$B_FIPL(R5)          ; Sync access to UCB
                  FEF3    30  2627  7381          BSBW    SAV_MULTI               ; Save the multicast addresses
                0240 8F   BB  262A  7382          PUSHR   #^M<R6,R9>              ; Save registers
                56   53   C2  262E  7383          SUBL    R3,R6                   ; Backup pointer to start of list
                59   53   D0  2631  7384          MOVL    R3,R9                   ; Setup string count in R9
                  0279    30  2634  7385          BSBW    SET_MULTI               ; See if we can set new addresses
                          2637  7386                                          ; R0 = return status
                0240 8F   BA  2637  7387          POPR    #^M<R6,R9>              ; Restore registers
                  FEF5    30  263B  7388          BSBW    RES_MULTI               ; Restore the multicast list
                          263E  7389          ENBINT                          ; Restore IPL
                38 50     E9  2641  7390          BLBC    R0,170$                 ; Br if error
                     12   11  2644  7391          BRB     130$                    ; Check if state okay
                          2646  7392
        05 59    00   E1  2646  7393  100$:    BBC     #PRM_FLG_V_MIN,R9,110$  ; Br if no minimum value
                87   53   B1  264A  7394          CMPW    R3,(R7)+                ; Is the value too small?
                     2D   1F  264D  7395          BLSSU   170$                    ; Br if yes - error
        05 59    01   E1  264F  7396  110$:    BBC     #PRM_FLG_V_MAX,R9,130$  ; Br if no maximum value
                87   53   B1  2653  7397          CMPW    R3,(R7)+                ; Is the value too big?
                     24   1A  2656  7398          BGTRU   170$                    ; Br if yes - error
        18 59    02   E1  2658  7399  130$:    BBC     #PRM_FLG_V_INVALID,R9,140$ ; Br if no invalid flags
                52   87   B0  265C  7400          MOVW    (R7)+,R2                ; Get invalid flags
        06 59    03   E0  265F  7401          BBS     #PRM_FLG_V_CDB,R9,135$  ; Br if CDB datum
        68 A5    52   B3  2663  7402          BITW    R2,UCB$W_DEVSTS(R5)     ; Check UCB invalid bits
                     09   11  2667  7403          BRB     137$                    ; Continue
                54   D5  2669  7404  135$:    TSTL    R4                      ; Is CDB present?
                     07   13  266B  7405          BEQL    140$                    ; Br if no - okay
           0246 C4   52   93  266D  7406          BITB    R2,CDB_B_STS(R4)        ; Check CDB invalid bits
                     08   12  2672  7407  137$:    BNEQ    170$                    ; Br on error
                  FEEB    31  2674  7408  140$:    BRW     30$                     ; Loop if more parameters
                          2677  7409
                50   01   9A  2677  7410  150$:    MOVZBL  S^#SS$_NORMAL,R0        ; Set success return
                     03   11  267A  7411          BRB     180$                    ; And return
                          267C  7412
                50   14   9A  267C  7413  170$:    MOVZBL  S^#SS$_BADPARAM,R0      ; Set error return
                03DC 8F   BA  267F  7414  180$:    POPR    #^M<R2,R3,R4,R6,R7,R8,R9> ; Restore registers
                          05  2683  7415          RSB                             ; Return to caller
```

```
                                2684  7417          .SBTTL  CHANGE_PARAM - UPDATE UCB/CDB BASED ON P2 BUFFER PARAMETERS
                                2684  7418
                                2684  7419 ;++
                                2684  7420 ; CHANGE_PARAM - Update UCB/CDB with P2 buffer parameters
                                2684  7421 ;
                                2684  7422 ; This routine is called to update the UCB/CDB with the P2 buffer parameters.
                                2684  7423 ; The parameters are stored in the appropriate cells of the UCB/CDB.
                                2684  7424 ; This routine can only modify the LINE PARAMETERS.
                                2684  7425 ;
                                2684  7426 ; Inputs:
                                2684  7427 ;
                                2684  7428 ;       R2 = Address of verification table
                                2684  7429 ;       R3 = IRP address
                                2684  7430 ;       R5 = UCB address
                                2684  7431 ;
                                2684  7432 ;       IPL = FIPL
                                2684  7433 ;
                                2684  7434 ; Outputs:
                                2684  7435 ;
                                2684  7436 ;       R0 = destroyed.
                                2684  7437 ;       All other registers are preserved.
                                2684  7438 ;
                                2684  7439 ;--
                                2684  7440
                                2684  7441 CHANGE_PARAM:                            ; Change the UCB/CDB parameters
      07DE 8F     BB            2684  7442          PUSHR   #^M<R1,R2,R3,R4,R6,R7,R8,R9,R10> ; Save registers
      5A   52     D0            2688  7443          MOVL    R2,R10                  ; Save table address
   56   2C A3     D0            268B  7444          MOVL    IRP$L_SVAPTE(R3),R6     ; Get system P2 buffer address
           03     12            268F  7445          BNEQ    5$                      ; Br if system buffer
         00D8     31            2691  7446 3$:     BRW     120$                    ; Else, return
                                2694  7447
:RNG0001           00C8 C5     D0  2694    .1 5$:     MOVL    UCB$L_XQ_CDB(R5),R4     ; Get CDB address
-2         56     66     D0      2699  7450          MOVL    P2B_L_POINTER(R6),R6    ; Point to start of data
      58   32 A3     3C          269C  7451          MOVZWL  IRP$W_BCNT(R3),R8       ; Get size of P2 buffer
                                26A0  7452 ;
                                26A0  7453 ; Loop to get next parameter from P2 buffer
                                26A0  7454 ;
      58   02     C2            26A0  7455 10$:    SUBL    #2,R8                   ; Try to get next parameter
           EC     19            26A3  7456          BLSS    3$                      ; Br if not there
      50   86     3C            26A5  7457          MOVZWL  (R6)+,R0                ; Get parameter type from P2
           79     13            26A8  7458          BEQL    90$                     ; Br if null value parameter
      57   5A     D0            26AA  7459          MOVL    R10,R7                  ; Get verification table address
0B0E  8F   50     B1            26AD  7460          CMPW    R0,#NMA$C_PCLI_PTY      ; Is this the protocol type?
           04     12            26B2  7461          BNEQ    20$                     ; Br if not
           04     A8            26B4  7462          BISW    #UCB$M_XQ_PROTYP,-      ; Indicate that protocol type specified
         68 A5                   26B6  7463                  UCB$W_DEVSTS(R5)        ;
                                26B8  7464 ;
                                26B8  7465 ; Loop to store buffer parameter in UCB/CDB
                                26B8  7466 ;
                                26B8  7467          ASSUME  PRM_W_TYPE EQ 0
      51   87     3C            26B8  7468 20$:    MOVZWL  (R7)+,R1                ; Get parameter type code
           66     13            26BB  7469          BEQL    90$                     ; Br if end of verify table
51    F000 8F     AA            26BD  7470          BICW    #^C<PRM_TYP_M_CODE>,R1  ; Clear all but type code
                                26C2  7471          ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
      59   87     9A            26C2  7472          MOVZBL  (R7)+,R9                ; Get flags byte
      51   50     B1            26C5  7473          CMPW    R0,R1                   ; Parameters match?
           17     13            26C8  7474          BEQL    30$                     ; Br if yes
```

```
                57   02   C0  26CA  7475          ADDL    #2,R7                           ; Skip offset word
                                26CD  7476          SKIP    PRM_FLG_V_MIN,R9,R7     ; Skip minimum value
                                26D3  7477          SKIP    PRM_FLG_V_MAX,R9,R7     ; Skip maximum value
                                26D9  7478          SKIP    PRM_FLG_V_INVALID,R9,R7 ; Skip invalid flags
                     D7   11    26DF  7479          BRB     20$                     ; Try next parameter
                                26E1  7480 ;
                                26E1  7481 ; Match found - nullify if same value & check min,max,valid,invalid
                                26E1  7482 ;
                51   87   B0  26E1  7483 30$:     MOVW    (R7)+,R1                ; Get offset + width
                     0A   EF  26E4  7484          EXTZV   #PRM_OFF_V_WIDTH,-      ; Get width only
              52     51   06  26E6  7485                  #PRM_OFF_S_WIDTH,R1,R2  ;  ...
                     00   EF  26E9  7486          EXTZV   #PRM_OFF_V_VALUE,-      ; Get offset only
              51     51   0A  26EB  7487                  #PRM_OFF_S_VALUE,R1,R1  ;  ...
                05   59   03  E0  26EE  7488          BBS     #PRM_FLG_V_CDB,R9,40$   ; Br if CDB datum
                     51   55  C0  26F2  7489          ADDL    R5,R1                   ; Compute offset in UCB
                          03  11  26F5  7490          BRB     50$                     ; Continue
                     51   54  C0  26F7  7491 40$:     ADDL    R4,R1                   ; Compute offset in CDB
                                26FA  7492 50$:     ASSUME  PRM_B_FLAG EQ PRM_W_TYPE+2
             2E FB   A7   0C  E0  26FA  7493          BBS     #PRM_TYP_V_STRING,-5(R7),100$ ; Br if string data
                     58   04  C2  26FF  7494          SUBL    #4,R8                   ; Can we get value?
                          68  19  2702  7495          BLSS    120$                    ; Br if no - exit
                     53   86  D0  2704  7496          MOVL    (R6)+,R3                ; Get parameter value
                                2707  7497          CASE    R2,TYPE=B,LIMIT=#1,<-   ; Br to handler
                                2707  7498                  60$,-                   ;   Byte value
                                2707  7499                  70$,-                   ;   Word value
                                2707  7500                  80$>                    ;   Longword value
                                2711  7501 ;
                                2711  7502 ; Byte, word, longword value in structure
                                2711  7503 ;
                61   53   90  2711  7504 60$:     MOVB    R3,(R1)                 ; Store byte value
                   FF89   31  2714  7505 65$:     BRW     10$                     ; Check remainder
                61   53   B0  2717  7506 70$:     MOVW    R3,(R1)                 ; Store word value
                   FF83   31  271A  7507          BRW     10$                     ; Check remainder
                61   55   D0  271D  7508 80$:     MOVL    R5,(R1)                 ; Store longword value
                   FF7D   31  2720  7509          BRW     10$                     ; See if more left in P2 buffer
                                2723  7510 ;
                                2723  7511 ; Unknown or invalidated parameter
                                2723  7512 ;
                56   04   C0  2723  7513 90$:     ADDL    #4,R6                   ; Skip value parameter
                58   04   C2  2726  7514          SUBL    #4,R8                   ; Assume a value parameter
                     E9   14  2729  7515          BGTR    65$                     ; Br if more
                     3F   11  272B  7516          BRB     120$                    ; Else, all done
                                272D  7517 ;
                                272D  7518 ; String parameter in structure
                                272D  7519 ;
                58   02   C2  272D  7520 100$:    SUBL    #2,R8                   ; Can we get string?
                     3A   19  2730  7521          BLSS    120$                    ; Br if no - exit
                59   86   3C  2732  7522          MOVZWL  (R6)+,R9                ; Get string length
                58   59   C2  2735  7523          SUBL    R9,R8                   ; Can we read entire string?
                     32   19  2738  7524          BLSS    120$                    ; Br if no - exit
           50   0B21   8F   B1  273A  7525          CMPW    #NMA$C_PCLI_DES,R0      ; Is this the destination address?
                     05   12  273F  7526          BNEQ    101$                    ; Br if no
                  0244   30  2741  7527          BSBW    SET_DESAD               ; Else, set new destination address
                     20   11  2744  7528          BRB     110$                    ; Continue
           50   0B04   8F   B1  2746  7529 101$:    CMPW    #NMA$C_PCLI_PHA,R0      ; Is this the physical address?
                     05   12  274B  7530          BNEQ    103$                    ; Br if no
                  022D   30  274D  7531          BSBW    SET_PHYAD               ; Else set new physical address
```

```
                     14   11  2750  7532              BRB     110$                            ; Continue
50   0B0F            8F   B1  2752  7533  103$:       CMPW    #NMA$C_PCLI_MCA,R0              ; Is this the multicast address list?
                     05   12  2757  7534              BNEQ    105$                            ; Br if no
                   0154   30  2759  7535              BSBW    SET_MULTI                       ; Else, set up new UCB multicast list
                     08   11  275C  7536              BRB     110$                            ; Continue
                     3E   BB  275E  7537  105$:       PUSHR   #^M<R1,R2,R3,R4,R5>             ; Save registers
61   66              59   28  2760  7538              MOVC3   R9,(R6),(R1)                    ; Store string
                     3E   BA  2764  7539              POPR    #^M<R1,R2,R3,R4,R5>             ; Restore registers
     56              59   C0  2766  7540  110$:       ADDL    R9,R6                           ; Point past the string in P2 buffer
                   FF34   31  2769  7541              BRW     10$                             ; Try for more in P2 buffer
                          276C  7542
                   07DE   8F   BA  276C  7543  120$:       POPR    #^M<R1,R2,R3,R4,R6,R7,R8,R9,R10> ; Restore registers
                          05   2770  7544              RSB                                     ; Return to caller
```

```
                                2771  7546          .SBTTL  RETURN_P2,  Return UCB/CDB buffer parameters
                                2771  7547
                                2771  7548 ;++
                                2771  7549 ; RETURN_P2 - Return P2 buffer parameters
                                2771  7550 ;
                                2771  7551 ; This routine is called to return the UCB/CDB buffer parameters.
                                2771  7552 ;
                                2771  7553 ; Inputs:
                                2771  7554 ;
                                2771  7555 ;       R3 = IRP address
                                2771  7556 ;       R5 = UCB address
                                2771  7557 ;
                                2771  7558 ; Implicit inputs:
                                2771  7559 ;
                                2771  7560 ;       IRP$L_XQ_P2BUF(R3) = User P2 buffer address
                                2771  7561 ;       IRP$W_XQ_USERSIZ(R3) = User P2 buffer size
                                2771  7562 ;
                                2771  7563 ; Outputs:
                                2771  7564 ;
                                2771  7565 ;       R0 = Size of buffer returned
                                2771  7566 ;       All other registers are preserved.
                                2771  7567 ;
                                2771  7568 ;--
                                2771  7569
                                2771  7570 RETURN_P2::                                ; Return P2 buffer parameters
          03DE 8F     BB        2771  7571          PUSHR   #^M<R1,R2,R3,R4,R6,R7,R8,R9> ; Save registers
               50     D4        2775  7572          CLRL    R0                          ; Assume no P2 buffer given
     56   40 A3     D0        2777  7573          MOVL    IRP$L_XQ_P2BUF(R3),R6       ; Get user P2 buffer address
               03     12        277B  7574          BNEQ    5$                          ; Br if given
            00B7     31        277D  7575          BRW     70$                         ; Else, return
                               2780  7576
;RNG0001    54   00C8 C5     D0   2780    .1 5$:   MOVL    UCB$L_XQ_CDB(R5),R4         ; Get CDB address
-2           58   38 A3     3C   2785  7579        MOVZWL  IRP$W_XQ_USERSIZ(R3),R8     ; Get size of user buffer
                    56     DD   2789  7580        PUSHL   R6                          ; Save start of data address
                    53     DD   278B  7581        PUSHL   R3                          ; Save IRP address
               51   D8F0 CF     9E   278D  7582        MOVAB   LINE_PARAM,R1           ; Get address of verification talbe
                                2792  7583 ;
                                2792  7584 ; Loop to return next parameter
                                2792  7585 ;
                                2792  7586          ASSUME  PRM_W_TYPE EQ 0
                    57   81     B0   2792  7587 10$:    MOVW    (R1)+,R7                ; Get parameter type code
                         03     12   2795  7588        BNEQ    11$                     ; Br if end of verify table
                      0096      31   2797  7589        BRW     65$                     ; ...
                                279A  7590
               57   F000 8F     AB   279A  7591 11$:    BICW3   #^C<PRM_TYP_M_CODE>,R7,R9 ; Get only the type code
                         59          279F
                    53   81     9A   27A0  7592        MOVZBL  (R1)+,R3                ; Get flags byte
                    50   81     B0   27A3  7593        MOVW    (R1)+,R0                ; Get offset + width
                                27A6  7594 ;
                                27A6  7595 ; We will only return NMA$C_PCLI_DES to the SHARED-LIMITED users.
                                27A6  7596 ;
                    59   0B21 8F     B1   27A6  7597        CMPW    #NMA$C_PCLI_DES,R9      ; Is this a point-to-point parameter?
                              07     12   27AB  7598        BNEQ    13$                     ; Br if not
                    00D8 C5   02     91   27AD  7599        CMPB    #NMA$C_ACC_LIM,UCB$B_XQ_ACC(R5) ; Is this a SHARED-LIMITED user?
                              43     12   27B2  7600        BNEQ    50$                     ; Br if not, else return parameter
                              0A     EF   27B4  7601 13$:    EXTZV   #PRM_OFF_V_WIDTH,-      ; Get width only
                    52   50   06          27B6  7602                #PRM_OFF_S_WIDTH,R0,R2  ;       ...
```

```
                00   EF   27B9  7603          EXTZV   #PRM_OFF_V_VALUE,-          ; Get offset only
          50    50   0A        27BB  7604                  #PRM_OFF_S_VALUE,R0,R0      ;
            05  53   03   E0   27BE  7605          BBS     #PRM_FLG_V_CDB,R3,15$       ; Br if CDB datum
                50   55   C0   27C2  7606          ADDL    R5,R0                       ; Compute offset in UCB
                     07   11   27C5  7607          BRB     17$                         ; Continue
                     54   D5   27C7  7608 15$:     TSTL    R4                          ; Is CDB given?
                     2C   13   27C9  7609          BEQL    50$                         ; Br if no
                50   54   C0   27CB  7610          ADDL    R4,R0                       ; Compute offset in CDB
                58   02   C2   27CE  7611 17$:     SUBL    #2,R8                       ; Any room left in buffer?
                     54   19   27D1  7612          BLSS    60$                         ; Br if no - all done
                86   57   B0   27D3  7613          MOVW    R7,(R6)+                    ; Return parameter
          30 FB A1   0C   E0   27D6  7614          BBS     #PRM_TYP_V_STRING,-5(R1),55$ ; Br if string parameter
                58   04   C2   27DB  7615          SUBL    #4,R8                       ; Any room left?
                     47   19   27DE  7616          BLSS    60$                         ; Br if no - all done
                          27E0  7617          CASE    R2,TYPE=B,LIMIT=#1,<-       ; Br to handler
                          27E0  7618                  20$,-                       ;   Byte value
                          27E0  7619                  30$,-                       ;   Word value
                          27E0  7620                  40$>                        ;   Longword value
                          27EA  7621 ;
                          27EA  7622 ; Byte, word, longword value in structure
                          27EA  7623 ;
                86   60   9A   27EA  7624 20$:     MOVZBL  (R0),(R6)+                  ; Store byte value
                     08   11   27ED  7625          BRB     50$                         ;
                86   60   3C   27EF  7626 30$:     MOVZWL  (R0),(R6)+                  ; Store word value
                     03   11   27F2  7627          BRB     50$                         ;
                86   60   D0   27F4  7628 40$:     MOVL    (R0),(R6)+                  ; Store longword value
                          27F7  7629 50$:     SKIP    PRM_FLG_V_MIN,R3,R1         ; Skip minimum value
                          27FD  7630          SKIP    PRM_FLG_V_MAX,R3,R1         ; Skip maximum value
                          2803  7631          SKIP    PRM_FLG_V_INVALID,R3,R1     ; Skip invalid flags
                     87   11   2809  7632          BRB     10$                         ; Try for more parameters
                          280B  7633 ;
                          280B  7634 ; String value in structure
                          280B  7635 ;
          59  0B0F  8F   B1   280B  7636 55$:     CMPW    #NMA$C_PCLI_MCA,R9          ; Is this the multicast address list?
                     05   12   2810  7637          BNEQ    57$                         ; Br if no
                   01C6   30   2812  7638          BSBW    RETURN_MULTI                ; Else, return multicast address list
                     E0   11   2815  7639          BRB     50$                         ; Try for more parameters
                58   08   C2   2817  7640 57$:     SUBL    #8,R8                       ; Any room left?
                     0B   19   281A  7641          BLSS    60$                         ; Br if no - all done
                86   06   9B   281C  7642          MOVZBW  #6,(R6)+                    ; Store string size
                86   80   D0   281F  7643          MOVL    (R0)+,(R6)+                 ; Move data
                86   80   B0   2822  7644          MOVW    (R0)+,(R6)+                 ;   ...
                     D0   11   2825  7645          BRB     50$                         ; Try for more parameters
                          2827  7646
                53   6E   D0   2827  7647 60$:     MOVL    (SP),R3                     ; Get IRP address
                0601 8F   B0   282A  7648          MOVW    #SS$_BUFFEROVF,IRP$W_XQ_STATUS(R3) ; Return error status
                  3A A3        282E
                     53 8ED0   2830  7649 65$:     POPL    R3                          ; Pop stack
          50    56   8E   C3   2833  7650          SUBL3   (SP)+,R6,R0                 ; Return size of parameters
                03DE 8F   BA   2837  7651 70$:     POPR    #^M<R1,R2,R3,R4,R6,R7,R8,R9> ; Restore registers
                          05   283B  7652          RSB                                 ; Return to caller
```

```
                                G 4
                                                .SBTTL  VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST

                                ;++
                                ; VALID_MULTI - VALIDATE THE MULTICAST ADDRESS LIST
                                ;
                                ; Functional description:
                                ;
                                ; This routine checks all address in the multicast address list to make sure
                                ; that the logical address bit (lsb) is on.
                                ;
                                ; Inputs:
                                ;
                                ;       R3 = Size of multicast string list
                                ;       R4 = CDB address
                                ;       R5 = UCB address
                                ;       R6 = Address past multicast strings
                                ;
                                ; Outputs:
                                ;
                                ;       R0 = Low bit clear if invalid address in list
                                ;       All other registers are preserved.
                                ;
                                ;--

                                VALID_MULTI:                            ; Validate the multicast address list
004C 8F     BB                  PUSHR   #^M<R2,R3,R6>                   ; Save some registers
     50 01  CE                  MNEGL   #1,R0                           ; Assume success
     56 53  C2                  SUBL    R3,R6                           ; Point back at start of list
     53 02  C2                  SUBL    #2,R3                           ; Can we read modifier word?
        26  19                  BLSS    20$                             ; Br if no - error
                                ;
                                ; Make sure modifier word is valid - non-zero and less than or equal to
                                ; NMA$C_LINMC_CAL
                                ;
                                        ASSUME  NMA$C_LINMC_SET EQ 1
                                        ASSUME  NMA$C_LINMC_CLR EQ 2
                                        ASSUME  NMA$C_LINMC_CAL EQ 3
52      86  B0                  MOVW    (R6)+,R2                        ; Get modifier value
        21  13                  BEQL    20$                             ; Br if zero - illegal
03      52  B1                  CMPW    R2,#NMA$C_LINMC_CAL             ; Is the modifier okay?
        1C  1A                  BGTRU   20$                             ; Br if no - error
        1C  13                  BEQL    30$                             ; Br if "CLEAR ALL" - ignore strings
53      06  C6                  DIVL    #6,R3                           ; Calculate number of strings
        17  13                  BEQL    30$                             ; Br if none
   12   66  E9                  BLBC    (R6),20$                        ; Br if not a logical address
50      86  D1                  CMPL    (R6)+,R0                        ; Do low order 32 bits = -1?
        05  12                  BNEQ    15$                             ; Br if no - okay
50      66  B1                  CMPW    (R6),R0                         ; Do high order 16 bits = -1?
        08  13                  BEQL    20$                             ; Br if yes - illegal
56      02  C0                  ADDL    #2,R6                           ; Point to next multicast address
  ED    53  F5                  SOBGTR  R3,10$                          ; Loop if more
        02  11                  BRB     30$                             ; Exit with success
                                ;
        50  D4                  CLRL    R0                              ; Return error
004C 8F     BA                  POPR    #^M<R2,R3,R6>                   ; Restore registers
            05                  RSB
```

```
                     2878  7711          .SBTTL  VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
                     2878  7712
                     2878  7713 ;++
                     2878  7714 ; VALID_PHYAD - VALIDATE THE PHYSICAL ADDRESS
                     2878  7715 ;
                     2878  7716 ; Functional description:
                     2878  7717 ;
                     2878  7718 ; This routine checks the physical address to make sure the LSB is clear and
                     2878  7719 ; that the modifier word is valid.
                     2878  7720 ;
                     2878  7721 ; Inputs:
                     2878  7722 ;
                     2878  7723 ;       R1 = Parameter type code
                     2878  7724 ;       R3 = Size of string
                     2878  7725 ;       R4 = CDB address
                     2878  7726 ;       R5 = UCB address
                     2878  7727 ;       R6 = Address past physical address string
                     2878  7728 ;
                     2878  7729 ; Outputs:
                     2878  7730 ;
                     2878  7731 ;       R0 = Low bit clear if invalid address in list
                     2878  7732 ;       All other registers are preserved.
                     2878  7733 ;
                     2878  7734 ;--
                     2878  7735
                     2878  7736 VALID_PHYAD:                          ; Validate the physical address
004C 8F   BB         2878  7737          PUSHR   #^M<R2,R3,R6>       ; Save some registers
50   01   CE         287C  7738          MNEGL   #1,R0               ; Assume success
56   53   C2         287F  7739          SUBL    R3,R6               ; Point back at start of list
53   02   C2         2882  7740          SUBL    #2,R3               ; Can we read modifier word?
     22   19         2885  7741          BLSS    30$                 ; Br if no - error
                     2887  7742
                     2887  7743 ;
                     2887  7744 ; Make sure modifier word is valid.
                     2887  7745 ;
                     2887  7746
                     2887  7747          ASSUME  NMA$C_LINMC_SET EQ 1
                     2887  7748          ASSUME  NMA$C_LINMC_CLR EQ 2
                     2887  7749          ASSUME  NMA$C_LINMC_CAL EQ 3
                     2887  7750          ASSUME  NMA$C_LINMC_SDF EQ 4
52   86   3C         2887  7751          MOVZWL  (R6)+,R2            ; Get modifier value
                     288A  7752          $DISPATCH       R2,TYPE=B,- ; Dispatch on modifier value
                     288A  7753                  <-
                     288A  7754                  <NMA$C_LINMC_SET 20$>,- ; Set the address
                     288A  7755                  <NMA$C_LINMC_CLR 40$>,- ; Clear the address
                     288A  7756                  <NMA$C_LINMC_CAL 30$>,- ; 3 - invalid value
                     288A  7757                  <NMA$C_LINMC_SDF 10$>,- ; 4 - check it out more
                     288A  7758                  >
     11   11         2896  7759          BRB     30$                 ; Any other values are invalid
                     2898  7760
51   0B04 8F   B1    2898  7761 10$:     CMPW    #NMA$C_PCLI_PHA,R1  ; Set to def physical addr requested?
     0A   12         289D  7762          BNEQ    30$                 ; Return failure if not
     0A   11         289F  7763          BRB     40$                 ; Else, success
                     28A1  7764
53   06   D1         28A1  7765 20$:     CMPL    #6,R3               ; Is string size okay?
     03   12         28A4  7766          BNEQ    30$                 ; Br if not
  02 66   E9         28A6  7767          BLBC    (R6),40$            ; Br if a physical address
```

```
                        28A9  7768
          50    D4      28A9  7769 30$:    CLRL    R0                      ; Return error
004C      8F    BA      28AB  7770 40$:    POPR    #^M<R2,R3,R6>           ; Restore registers
                05      28AF  7771         RSB
```

```
                     28B0  7773          .SBTTL  SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
                     28B0  7774
                     28B0  7775 ;++
                     28B0  7776 ; SET_MULTI - SET THE UCB MULTICAST ADDRESS LIST
                     28B0  7777 ;
                     28B0  7778 ; Functional description:
                     28B0  7779 ;
                     28B0  7780 ; This routine sets up the multicast addresses in the UCB.
                     28B0  7781 ;
                     28B0  7782 ; Inputs:
                     28B0  7783 ;
                     28B0  7784 ;       R4 = CDB address
                     28B0  7785 ;       R5 = UCB address
                     28B0  7786 ;       R6 = Address of multicast addresses to be set or cleared
                     28B0  7787 ;       R9 = Size of multicast list
                     28B0  7788 ;
                     28B0  7789 ;       IPL = FIPL
                     28B0  7790 ;
                     28B0  7791 ; Outputs:
                     28B0  7792 ;
                     28B0  7793 ;       R0 = Status return for request
                     28B0  7794 ;       All registers are preserved.
                     28B0  7795 ;
                     28B0  7796 ;--
                     28B0  7797
                     28B0  7798 SET_MULTI:                              ; Set up the UCB multicast address list
 024E 8F    BB       28B0  7799          PUSHR   #^M<R1,R2,R3,R6,R9>    ; Save registers
 59   02    C2       28B4  7800          SUBL    #2,R9                  ; Can we read the modifier word?
      65    19       28B7  7801          BLSS    90$                    ; Br if no - exit
 59   06    C6       28B9  7802          DIVL    #6,R9                  ; Calculate number of addresses
 51   86    3C       28BC  7803          MOVZWL  (R6)+,R1               ; Get the modifier
                     28BF  7804          ASSUME  NMA$C_LINMC_SET EQ 1
                     28BF  7805          ASSUME  NMA$C_LINMC_CLR EQ 2
                     28BF  7806          ASSUME  NMA$C_LINMC_CAL EQ 3
                     28BF  7807          CASE    R1,TYPE=B,LIMIT=#1,<-  ; Dispatch on modifier
                     28BF  7808                  10$,-                  ; Set the address(es)
                     28BF  7809                  40$,-                  ; Clear the address(es)
                     28BF  7810                  70$>                   ; Clear ALL addresses
                     28C9  7811 ;
                     28C9  7812 ; Set address from list
                     28C9  7813 ;
      59    D5       28C9  7814 10$:     TSTL    R9                     ; Any addresses present?
      51    13       28CB  7815          BEQL    90$                    ; Br if no - exit
 51   86    D0       28CD  7816 20$:     MOVL    (R6)+,R1               ; Get multicast address
 52   86    B0       28D0  7817          MOVW    (R6)+,R2               ;       ...
    0168    30       28D3  7818          BSBW    MATCH_ADDRESS          ; Try to find address in table
   0F 50    E8       28D6  7819          BLBS    R0,30$                 ; Br if present - skip it
      4B    10       28D9  7820          BSBB    FIND_MLTENTRY          ; Find entry in UCB multicast table
   43 50    E9       28DB  7821          BLBC    R0,100$                ; Br if none - leave in error
 83   51    D0       28DE  7822          MOVL    R1,(R3)+               ; Insert new address
 83   52    B0       28E1  7823          MOVW    R2,(R3)+               ;       ...
 00E9 C5    96       28E4  7824          INCB    UCB$B_XQ_MULTI(R5)     ; Count one more address
   E2 59    F5       28E8  7825 30$:     SOBGTR  R9,20$                 ; Br if more
      31    11       28EB  7826          BRB     90$                    ; All done
                     28ED  7827 ;
                     28ED  7828 ; Clear address from list
                     28ED  7829 ;
```

```
                     59   D5  28ED  7830 40$:     TSTL    R9                          ; Any addresses present?
                     2D   13  28EF  7831          BEQL    90$                         ; Br if no - exit
                51   86   D0  28F1  7832 50$:     MOVL    (R6)+,R1                    ; Get multicast address
                52   86   B0  28F4  7833          MOVW    (R6)+,R2                    ;                   ...
                   0144   30  28F7  7834          BSBW    MATCH_ADDRESS               ; Try to find address in table
                  0B 50   E9  28FA  7835          BLBC    R0,60$                      ; Br if not present - skip it
                     83   D4  28FD  7836          CLRL    (R3)+                       ; Mark slot as not in use
                     83   B4  28FF  7837          CLRW    (R3)+                       ;                   ...
                00E9 C5   97  2901  7838          DECB    UCB$B_XQ_MULTI(R5)          ; Count one less address
                   0042   30  2905  7839          BSBW    SQUEEZ_MULTI                ; Squeeze up the multicast list
                  E6 59   F5  2908  7840 60$:     SOBGTR  R9,50$                      ; Br if more
                     11   11  290B  7841          BRB     90$                         ; All done
                          290D  7842 ;
                          290D  7843 ; Clear all multicast addresses
                          290D  7844 ;
                00E9 C5   94  290D  7845 70$:     CLRB    UCB$B_XQ_MULTI(R5)          ; Reset number of multicast addresses
                51   24   9A  2911  7846          MOVZBL  #MAX_C_MCT*3,R1             ; Get number of words in multicast list
             52 00EC C5   9E  2914  7847          MOVAB   UCB$G_XQ_MULTI(R5),R2       ; Get address of multicast addresses
                     82   B4  2919  7848 80$:     CLRW    (R2)+                       ; Init multicast address list
                  FB 51   F5  291B  7849          SOBGTR  R1,80$                      ; Loop if more
                          291E  7850
                50   01   9A  291E  7851 90$:     MOVZBL  S^#SS$_NORMAL,R0            ; Return success
                024E 8F   BA  2921  7852 100$.    POPR    #^M<R1,R2,R3,R6,R9>         ; Restore registers
                          05  2925  7853          RSB                                 ; Return to caller
                          2926  7854
                          2926  7855 ;+
                          2926  7856 ; FIND_MLTENTRY - FIND EMPTY SLOT IN UCB MULTICAST ADDRESS LIST
                          2926  7857 ;
                          2926  7858 ; Inputs:
                          2926  7859 ;
                          2926  7860 ;       R5 = UCB address
                          2926  7861 ;
                          2926  7862 ; Outputs:
                          2926  7863 ;
                          2926  7864 ;       R0 = Status return for request
                          2926  7865 ;       R3 = Address of available slot if successful
                          2926  7866 ;-
                          2926  7867 FIND_MLTENTRY:
                     51   DD  2926  7868          PUSHL   R1                          ; Save R1
                50   01   CE  2928  7869          MNEGL   #1,R0                       ; Assume success
             53 00EC C5   9E  292B  7870          MOVAB   UCB$G_XQ_MULTI(R5),R3       ; Get address of multicast list
                51   0C   9A  2930  7871          MOVZBL  #MAX_C_MCT,R1               ; Get maximum number of addresses
                     83   D5  2933  7872 10$:     TSTL    (R3)+                       ; Empty slot?
                     04   12  2935  7873          BNEQ    20$                         ; Br if no - skip to next entry
                     63   B5  2937  7874          TSTW    (R3)                        ; Really?
                     08   13  2939  7875          BEQL    30$                         ; Br if yes - success
                53   02   C0  293B  7876 20$:     ADDL    #2,R3                       ; Skip to next address
                  F2 51   F5  293E  7877          SOBGTR  R1,10$                      ; Br if more to try
                          2941  7878
                     50   D4  2941  7879          CLRL    R0                          ; Return failure
                53   04   C2  2943  7880 30$:     SUBL    #4,R3                       ; Back up pointer
                     51 8ED0  2946  7881          POPL    R1                          ; Restore registers
                          05  2949  7882          RSB
                          294A  7883
                          294A  7884 ;+
                          294A  7885 ; SQUEEZ_MULTI - SQUEEZE UP THE MULTICAST ADDRESS LIST
                          294A  7886 ;
```

```
                     294A  7887 ; Functional description:
                     294A  7888 ;
                     294A  7889 ; The first empty slot is searched for in the multicast address list and
                     294A  7890 ; the list is squeezed from that point down.
                     294A  7891 ;
                     294A  7892 ; Inputs:
                     294A  7893 ;
                     294A  7894 ;       R5 = UCB address
                     294A  7895 ;
                     294A  7896 ; Ouputs:
                     294A  7897 ;
                     294A  7898 ;       None
                     294A  7899 ;
                     294A  7900 ;--
                     294A  7901 SQUEEZ_MULTI:                                ; Squeeze down the multicast address list
                     294A  7902         PUSHQ   R3                           ; Save R3, R4
54   00E9 C5   9A    294D  7903         MOVZBL  UCB$B_XQ_MULTI(R5),R4        ; Get number of entries in list
           25  13    2952  7904         BEQL    40$                          ; Br if none
53   00EC C5   9E    2954  7905         MOVAB   UCB$G_XQ_MULTI(R5),R3        ; Get address of multicast list
                     2959  7906 ;
                     2959  7907 ; Search for empty slot in list
                     2959  7908 ;
           63  D5    2959  7909 10$:    TSTL    (R3)                         ; Is this the empty slot
           05  12    295B  7910         BNEQ    20$                          ; Br if no
        04 A3  B5    295D  7911         TSTW    4(R3)                        ; Really?
           08  13    2960  7912         BEQL    30$                          ; Br if yes
     53    06  C0    2962  7913 20$:    ADDL    #6,R3                        ; Skip to next entry in list
        F1 54  F5    2965  7914         SOBGTR  R4,10$                       ; Loop if more possiblities
           0F  11    2968  7915         BRB     40$                          ; Else, exit
                     296A  7916 ;
                     296A  7917 ; Empty slot found - put the squeeze on.
                     296A  7918 ;
83   06 A3   D0      296A  7919 30$:    MOVL    6(R3),(R3)+                  ; Squeeze the list
83   06 A3   B0      296E  7920         MOVW    6(R3),(R3)+                  ;       ...
     F5 54   F5      2972  7921         SOBGTR  R4,30$                       ; Loop if more to go
        83   D4      2975  7922         CLRL    (R3)+                        ; Zero the last entry
        83   B4      2977  7923         CLRW    (R3)+                        ;       ...
                     2979  7924 40$:    POPQ    R3                           ; Restore R3, R4
                 05  297C  7925         RSB                                  ; Return to caller
```

```
                     297D  7927          .SBTTL  SET_PHYAD - SET THE PHYSICAL ADDRESS
                     297D  7928          .SBTTL  SET_DESAD - SET THE DESTINATION ADDRESS
                     297D  7929 ;++
                     297D  7930 ; SET_PHYAD - SET THE PHYSICAL ADDRESS
                     297D  7931 ; SET_DESAD - SET THE DESTINATION ADDRESS
                     297D  7932 ;
                     297D  7933 ; Functional description:
                     297D  7934 ;
                     297D  7935 ; This routine sets up the physical address in the CDB.
                     297D  7936 ;
                     297D  7937 ; Inputs:
                     297D  7938 ;
                     297D  7939 ;       R4 = CDB address
                     297D  7940 ;       R5 = UCB address
                     297D  7941 ;       R6 = Address of physical address to be set or cleared
                     297D  7942 ;       R9 = Size of the string
                     297D  7943 ;
                     297D  7944 ;       IPL = FIPL
                     297D  7945 ;
                     297D  7946 ; Outputs:
                     297D  7947 ;
                     297D  7948 ;       R0 = destroyed.
                     297D  7949 ;       All other registers are preserved.
                     297D  7950 ;
                     297D  7951 ;--
                     297D  7952          .ENABL  LSB
                     297D  7953 SET_PHYAD:                                   ; Set up the CDB physical address
     02C2 8F    BB   297D  7954          PUSHR   #^M<R1,R6,R7,R9>           ; Save registers
57   00E2 C5    9E   2981  7955          MOVAB   UCB$G_XQ_PHA(R5),R7        ; Get address of UCB cell
          09    11   2986  7956          BRB     10$                        ; Join common code
                     2988  7957 SET_DESAD:
     02C2 8F    BB   2988  7958          PUSHR   #^M<R1,R6,R7,R9>           ; Save registers
57   00D0 C5    9E   298C  7959          MOVAB   UCB$G_XQ_DES(R5),R7        ; Get address of UCB cell
59   02    C2   2991  7960 10$:    SUBL    #2,R9                      ; Can we read the modifier word?
     40    19   2994  7961          BLSS    50$                        ; Br if no - exit
59   06    C6   2996  7962          DIVL    #6,R9                      ; Calculate number of addresses
51   86    3C   2999  7963          MOVZWL  (R6)+,R1                   ; Get the modifier
                     299C  7964
                     299C  7965          ASSUME  NMA$C_LINMC_SET EQ 1
                     299C  7966          ASSUME  NMA$C_LINMC_CLR EQ 2
                     299C  7967          ASSUME  NMA$C_LINMC_CAL EQ 3
                     299C  7968          ASSUME  NMA$C_LINMC_SDF EQ 4
                     299C  7969          $DISPATCH       R1,TYPE=B,-        ; Dispatch on modifier value
                     299C  7970                  <-
                     299C  7971                  <NMA$C_LINMC_SET 20$>,- , Set the address
                     299C  7972                  <NMA$C_LINMC_CLR 30$>,- ; Clear the address
                     299C  7973                  <NMA$C_LINMC_CAL 50$>,- ; 3 - invalid value
                     299C  7974                  <NMA$C_LINMC_SDF 40$>,- ; Set physical as DEFnet default address
                     299C  7975                  >
     2C    11   29A8  7976          BRB     50$                        ; Any other values are invalid
                     29AA  7977
                     29AA  7978 ;
                     29AA  7979 ; Set physical address
                     29AA  7980 ;
     59    D5   29AA  7981 20$:    TSTL    R9                         ; Any addresses present?
     28    13   29AC  7982          BEQL    50$                        ; Br if no - exit
87   86    D0   29AE  7983          MOVL    (R6)+,(R7)+                ; Set new address
```

```
                     67  86   B0  29B1  7984          MOVW    (R6)+,(R7)              ;                ...
                         20   11  29B4  7985          BPB     50$                     ; All done
                                  29B6  7986 ;
                                  29B6  7987 ; Clear address from list
                                  29B6  7988 ;
                     87  01   CE  29B6  7989 30$:      MNEGL   #1,(R7)+                ; Reset address
                     67  01   AE  29B9  7990          MNEGW   #1,(R7)                 ;                ...
                         18   11  29BC  7991          BRB     50$                     ; All done
                                  29BE  7992
                                  29BE  7993 ;
                                  29BE  7994 ; Set the physical address (Assume entered from SET_PHYAD) to the DECNET
                                  29BE  7995 ; default address. The PHA address has been pre-set with -1, which implies
                                  29BE  7996 ; using the HWA.
                                  29BE  7997 ;
;RNG0001                                 29BE    .1 40$:     ASSUME  CDB_STS_V_INITED  EQ  0
;RNG0001              13 0246 C4   E8  29BE    .2          BLBS    CDB_B_STS(R4),50$       ; Br if device already inited, use HWA
-2                00000000'GF   B0  29C3  8000          MOVW    G^SCS$GB_SYSTEMID,R1    ; Else, get SCSSYSTEMID parameter
                         51       29C9
                         0A   13  29CA  8001          BEQL    50$                     ; Br if none, use HWA
                    000400AA 8F   D0  29CC  8002          MOVL    #^X000400AA,(R7)+       ; Set common low longword of addr
                         87       29D2
                     67  51   B0  29D3  8003          MOVW    R1,(R7)                 ; Set unique high word of address
                                  29D6  8004
                     02C2 8F   BA  29D6  8005 50$:      POPR    #^M<R1,R6,R7,R9>        ; Restore registers
                         05       29DA  8006          RSB                             ; Return to caller
                                  29DB  8007          .DSABL  LSB
```

```
                        29DB  8009          .SBTTL  RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
                        29DB  8010
                        29DB  8011 ;++
                        29DB  8012 ; RETURN_MULTI - RETURN THE MULTICAST ADDRESS LIST
                        29DB  8013 ;
                        29DB  8014 ; Functional description:
                        29DB  8015 ;
                        29DB  8016 ; This routine returns all the multicast addresses in the UCB list.
                        29DB  8017 ;
                        29DB  8018 ; Inputs:
                        29DB  8019 ;
                        29DB  8020 ;       R4 = CDB address
                        29DB  8021 ;       R5 = UCB address
                        29DB  8022 ;       R6 = Address where to store multicast address list
                        29DB  8023 ;       R8 = Size left in output buffer
                        29DB  8024 ;
                        29DB  8025 ; Outputs:
                        29DB  8026 ;
                        29DB  8027 ;       R6 = Address past stored multicast address list
                        29DB  8028 ;       All other registers are preserved.
                        29DB  8029 ;
                        29DB  8030 ;--
                        29DB  8031
                        29DB  8032 RETURN_MULTI:                                ; Return the multicast address list
                        29DB  8033          PUSHQ   R1                          ; Save R1, R2
52   00E9 C5   9A       29DE  8034          MOVZBL  UCB$B_XQ_MULTI(R5),R2       ; Set number of multicast addresses
51   00EC C5   9E       29E3  8035          MOVAB   UCB$G_XQ_MULTI(R5),R1       ; Point to start of multicast lists
     58   02   C2       29E8  8036          SUBL    #2,R8                       ; Can string size fit?
          1E   19       29EB  8037          BLSS    40$                         ; Br if no - return
     56   02   C0       29ED  8038          ADDL    #2,R6                       ; Skip size field of return data
          56   DD       29F0  8039          PUSHL   R6                          ; Save start address of string
     58   06   C2       29F2  8040 10$:     SUBL    #6,R8                       ; Room left in buffer?
          09   19       29F5  8041          BLSS    30$                         ; Br if no .. exit now
     86   81   D0       29F7  8042          MOVL    (R1)+,(R6)+                 ; Return address
     86   81   B0       29FA  8043          MOVW    (R1)+,(R6)+                 ;       ...
       F2 52   F5       29FD  8044          SOBGTR  R2,10$                      ; Br if more possibles
                        2A00  8045
          51 8ED0       2A00  8046 30$:     POPL    R1                          ; Restore string address
52   56   51   C3       2A03  8047          SUBL3   R1,R6,R2                    ; Get string size
  FE A1   52   B0       2A07  8048          MOVW    R2,-2(R1)                   ; Return string size
                        2A0B  8049 40$:     POPQ    R1                          ; Restore R1, R2
                  05    2A0E  8050          RSB                                 ; Return to caller
```

```
                        2A0F  8052          .SBTTL  MATCH_MULTI - CHECK MULTICAST ADDRESS
                        2A0F  8053
                        2A0F  8054 ;++
                        2A0F  8055 ; MATCH_MULTI - CHECK MULTICAST ADDRESS
                        2A0F  8056 ;
                        2A0F  8057 ; Functional description:
                        2A0F  8058 ;
                        2A0F  8059 ; This routine returns success if the unit is in promiscous mode, or the
                        2A0F  8060 ; recognizes all multicast address or the multicast address in the buffer
                        2A0F  8061 ; matches a multicast address in the unit's multicast address list.
                        2A0F  8062 ;
                        2A0F  8063 ; Inputs:
                        2A0F  8064 ;
                        2A0F  8065 ;       R2 = Receive buffer
                        2A0F  8066 ;       R4 = CDB address
                        2A0F  8067 ;       R5 = UCB address
                        2A0F  8068 ;
                        2A0F  8069 ; Outputs:
                        2A0F  8070 ;
                        2A0F  8071 ;       R0 = Status return for request
                        2A0F  8072 ;       R1 = Destroyed
                        2A0F  8073 ;       All other registers are preserved.
                        2A0F  8074 ;
                        2A0F  8075 ;--
                        2A0F  8076
                        2A0F  8077 MATCH_MULTI:                             ; Find multicast addres in UCB
                        2A0F  8078          PUSHQ   R2                      ; Save R2, R3
                  50    D4  2A12  8079          CLRL    R0                      ; Assume failure
                        2A14  8080
                        2A14  8081          ASSUME  UCB$V_XQ_INITED  EQ  0
                22 68   A5  E9  2A14  8082          BLBC    UCB$W_DEVSTS(R5),10$    ; Br if PROTOCOL TYPE is not valid
                        50  D6  2A18  8083          INCL    R0                      ; Assume success
                        2A1A  8084          ASSUME  NMA$C_STATE_ON EQ 0
                        2A1A  8085          ASSUME  NMA$C_STATE_OFF EQ 1
              1B 00DE   C5  E9  2A1A  8086          BLBC    UCB$B_XQ_PRM(R5),10$    ; Br if promiscuous mode - OKAY
              16 00DF   C5  E9  2A1F  8087          BLBC    UCB$B_XQ_MLT(R5),10$    ; Br if all multicasts are enabled
              51    38  A2  7D  2A24  8088          MOVQ    CXB$G_R_DEST(R2),R1     ; Get multicast address
           FFFFFFFF   8F  D1  2A28  8089          CMPL    #BRDCST,R1              ; Is this the broadcast address?
                        51      2A2E
                        07  12  2A2F  8090          BNEQ    5$                      ; Br if no
            52  FFFF    8F  B1  2A31  8091          CMPW    #BRDCST2,R2             ; Really?
                        02  13  2A36  8092          BEQL    10$                     ; Br if yes - everybody get this one
                        04  10  2A38  8093  5$:     BSBB    MATCH_ADDRESS           ; Else, find exact match
                            2A3A  8094  10$:    POPQ    R2                      ; Restore R2, R3
                        05  2A3D  8095          RSB                             ; Return to caller
```

```
                     2A3E  8097          .SBTTL  MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
                     2A3E  8098
                     2A3E  8099 ;++
                     2A3E  8100 ; MATCH_ADDRESS - FIND A MATCH ON A MULTICAST ADDRESS
                     2A3E  8101 ;
                     2A3E  8102 ; Functional description:
                     2A3E  8103 ;
                     2A3E  8104 ; This routine searches the UCB multicast address list for a match on a
                     2A3E  8105 ; multicast address.
                     2A3E  8106 ;
                     2A3E  8107 ; Inputs:
                     2A3E  8108 ;
                     2A3E  8109 ;       R1 = Low 32 bits of 48 bit multicast address to match
                     2A3E  8110 ;       R2 = High 16 bits of 48 bit multicast address to match
                     2A3E  8111 ;       R4 = CDB address
                     2A3E  8112 ;       R5 = UCB address
                     2A3E  8113 ;
                     2A3E  8114 ; Outputs:
                     2A3E  8115 ;
                     2A3E  8116 ;       R0 = Status return for request
                     2A3E  8117 ;       R3 = Address of slot in multicast address list
                     2A3E  8118 ;       All other registers are preserved.
                     2A3E  8119 ;
                     2A3E  8120 ;--
                     2A3E  8121
                     2A3E  8122 MATCH_ADDRESS:                          ; Find multicast address in UCB
          54    DD   2A3E  8123          PUSHL   R4                     ; Save R4
     50   01    9A   2A40  8124          MOVZBL  S^#SS$_NORMAL,R0       ; Assume success
54   00E9  C5    9A   2A43  8125          MOVZBL  UCB$B_XQ_MULTI(R5),R4  ; Set number of multicast addresses
53   00EC  C5    9E   2A48  8126          MOVAB   UCB$G_XQ_MULTI(.5),R3  ; Point to start of multicast lists
     51    83    D1   2A4D  8127 10$:     CMPL    (R3)+,R1               ; Is this a match?
           05    12   2A50  8128          BNEQ    20$                    ; Br if no - skip to next
     52    63    B1   2A52  8129          CMPW    (R3),R2                ; Is it really?
           08    13   2A55  8130          BEQL    30$                    ; Br if yes - all done
     53    02    C0   2A57  8131 20$:     ADDL    #2,R3                  ; Skip to next entry
       F0  54    F5   2A5A  8132          SOBGTR  R4,10$                 ; Br if more in list
                    2A5D  8133
           50    D4   2A5D  8134          CLRL    R0                     ; Return failure
     53    04    C2   2A5F  8135 30$:     SUBL    #4,R3                  ; Backup pointer
           54 8ED0   2A62  8136          POPL    R4                     ; Restore R4
                 05   2A65  8137          RSB                            ; Return to caller
```

```
                                                E 5
                     2A66  8139          .SBTTL  POKE_USER - DELIVER ATTENTION ASTS
                     2A66  8140
                     2A66  8141 ;++
                     2A66  8142 ; POKE_USER - Deliver attention AST
                     2A66  8143 ;
                     2A66  8144 ; Functional description:
                     2A66  8145 ;
                     2A66  8146 ; This routine is used to deliver an attention AST if one has been
                     2A66  8147 ; requested.
                     2A66  8148 ;
                     2A66  8149 ; Inputs:
                     2A66  8150 ;
                     2A66  8151 ;       R5 = UCB address
                     2A66  8152 ;
                     2A66  8153 ; Outputs:
                     2A66  8154 ;
                     2A66  8155 ;       R0 = Low bit clear only if user is not notified
                     2A66  8156 ;       R1-R3 are destroyed.
                     2A66  8157 ;
                     2A66  8158 ;--
                     2A66  8159
                     2A66  8160 POKE_USER:                                  ; Poke user process
                     2A66  8161          DSBINT  UCB$B_FIPL(R5)             ; Sync access to UCB
           7E   D4   2A6D  8162          CLRL    -(SP)                      ; Assume failure
     51    00C0 C5   9E   2A6F  8163          MOVAB   UCB$L_XQ_AST(R5),R1        ; Get AST listhead
           61   D5   2A74  8164          TSTL    (R1)                       ; Empty?
           1C   13   2A76  8165          BEQL    30$                        ; Branch if yes
           6E   D6   2A78  8166          INCL    (SP)                       ; Indicate success
           54   DD   2A7A  8167          PUSHL   R4                         ; Save R4
     54    51   D0   2A7C  8168          MOVL    R1,R4                      ; Copy listhead address
     51    61   D0   2A7F  8169 10$:     MOVL    (R1),R1                    ; Address a block
           07   13   2A82  8170          BEQL    20$                        ; Branch if done
        44 A5   D0   2A84  8171          MOVL    UCB$L_DEVDEPEND(R5),-      ; Change parameter
        1C A1        2A87  8172                  ACB$L_KAST+4(R1)           ;    return status
           F4   11   2A89  8173          BRB     10$                        ; Continue thru AST blocks
00000000'GF   16   2A8B  8174 20$:     JSB     G^COM$DELATTNAST           ; Deliver the AST's
           54 8ED0   2A91  8175          POPL    R4                         ; Restore R4
                     2A94  8176
           50 8ED0   2A94  8177 30$:     POPL    R0                         ; Return success indicator
                     2A97  8178          ENBINT                             ; Restore IPL
                05   2A9A  8179          RSB                                ; Return to caller
```

```
                                2A9B  8181          .SBTTL  MATCH_PROTYP - Match protocol type
                                2A9B  8182          .SBTTL  MATCH_PROMTYP - Find the promiscuous user
                                2A9B  8183
                                2A9B  8184 ;++
                                2A9B  8185 ; MATCH_PROTYP - Match protocol type
                                2A9B  8186 ; MATCH_PROMTYP - Find the promiscuous user
                                2A9B  8187 ;
                                2A9B  8188 ; This routine checks for a match of a protocol type against that in
                                2A9B  8189 ; existing UCB's.
                                2A9B  8190 ;
                                2A9B  8191 ; Inputs:
                                2A9B  8192 ;
                                2A9B  8193 ;       R1 = word of protocol type
                                2A9B  8194 ;       R4 = CDB address
                                2A9B  8195 ;
                                2A9B  8196 ; Outputs:
                                2A9B  8197 ;
                                2A9B  8198 ;       R0 = LBS=> match; LBC=> no match
                                2A9B  8199 ;       R5 = UCB address on success
                                2A9B  8200 ;
                                2A9B  8201 ;-
                                2A9B  8202
                                2A9B  8203 MATCH_PROTYP:                            ; Match protocol type
                          50    D4  2A9B  8204          CLRL    R0                    ; Assume failure
                    55  0110 C4   D0  2A9D  8205          MOVL    CDB_L_UCB0(R4),R5     ; Get first UCB address
                          18    13  2AA2  8206          BEQL    20$                   ; Br if not inited - yet
                      55  30 A5   D0  2AA4  8207 10$:    MOVL    UCB$L_LINK(R5),R5     ; Get next UCB address
                          12    13  2AA8  8208          BEQL    20$                   ; If EQL no match
                                2AAA  8209          ASSUME  UCB$V_XQ_INITED  EQ  0
                        F6 68 A5    E9  2AAA  8210          BLBC    UCB$W_DEVSTS(R5),10$  ; Br if PROTOCOL TYPE is not valid
                                2AAE  8211
                                2AAE  8212          ASSUME  NMA$C_STATE_ON EQ 0
                                2AAE  8213          ASSUME  NMA$C_STATE_OFF EQ 1
                                2AAE  8214
                      F1 00DE C5    E9  2AAE  8215          BLBC    UCB$B_XQ_PRM(R5),10$  ; Skip if PROMISCUOUS user
                    00CE C5   51    B1  2AB3  8216          CMPW    R1,UCB$W_XQ_PROTYP(R5) ; Match?
                              EA    12  2AB8  8217          BNEQ    10$                   ; If NEQ no - loop
                              50    D6  2ABA  8218 15$:     INCL    R0                    ; Return success
                              05    2ABC  8219 20$:     RSB                           ; Done
                                2ABD  8220
                                2ABD  8221 MATCH_PROMTYP:
                          50    01  9A  2ABD  8222          MOVZBL  #1,R0                 ; Assume success
                    55  0210 C4   D0  2AC0  8223          MOVL    CDB_L_PRMUSER(R4),R5  ; Get PROMISCUOUS user's UCB address
                              02    12  2AC5  8224          BNEQ    10$                   ; Br if present
                              50    D4  2AC7  8225          CLRL    R0                    ; Else, return error
                              05    2AC9  8226 10$:     RSB                           ; Return to caller
                                2ACA  8227
                                2ACA  8228
                                2ACA  8229 XQ_END::
                                2ACA  8230          .END
```

```
$$$                     = 00000020 R      02
$$$FLG                  = 00000013
$$$MAP                  = 00000000
$$$OFF                  = 000000F3
$$$SIZ                  = 00000008
$$$TYP                  = 00000429
$$$WID                  = 00000002
$$BASE                  = 00000001
$$DISPL                 = 00000009
$$GENSW                 = 00000001
$$HIGH                  = 00000008
$$LIMIT                 = 00000007
$$LOW                   = 00000001
$$MNSW                  = 00000001
$$MXSW                  = 00000001
$$OP                    = 00000002
ABORTIO                   00000340 R      03
ABORTIO_BR                00000288 R      03
ABORT_IRP                 00000CC3 R      03
ABORT_PKT                 00001B39 R      03
ACB$L_KAST              = 00000018
ACCESS                    00000DDD R      03
ADDRCVLIST                00001435 RG     03
ADD_MULTI                 00002493 R      03
ALLOC_CDB                 00001EB3 R      03
ALLOC_P2BUF               00000DF6 R      03
ALT_START                 00000751 RG     03
ASSEM_PKTS                00001BA1 RG     03
ATS_UBA                 = 00000001
BAD_PARAM_TBL             000001A6 R      03
BLD_IRP                   00002275 R      03
BLD_STOP_IRP              0000223C R      03
BLK_B_SPARE               0000000B
BLK_B_TYPE                0000000A
BLK_C_HEADER              0000000C
BLK_L_LINK                00000000
BLK_T_DATA                0000000C
BLK_W_SIZE                00000008
BRDCST1                 = FFFFFFFF
BRDCST2                 = 0000FFFF
BUG$_NOBUFPCKT            ********    X     03
BUG$_UNSUPRTCPU           ********    X     03
CAN$C_CANCEL            = 00000000
CAN$C_DASSGN            = 00000001
CANCEL                    00002315 RG     03
CCB$L_UCB               = 00000000
CDB_B_AQUOTA              000002EE  G
CDB_B_CON                 00000249  G
CDB_B_DIAG1               00000119  G
CDB_B_FIPL                0000000B  G
CDB_B_LASTRCV             00000104  G
CDB_B_LASTXMT             00000105  G
CDB_B_MLT                 00000248  G
CDB_B_MLTTBL              0000025D  G
CDB_B_MQUOTA              000002EF  G
CDB_B_MULTI               0000025C  G
CDB_B_NEXTRCV             00000019  G
CDB_B_NEXTXMT                       00000018  G
CDB_B_PRM                           00000247  G
CDB_B_RCVCNT                        00000106  G
CDB_B_RCVMAP                        0000001A  G
CDB_B_SETPRM                        00000249  G
CDB_B_SPARE                         00000118  G
CDB_B_STS                           00000246  G
CDB_B_TIM_XMT                       0000020A  G
CDB_B_TYPE                          0000000A  G
CDB_B_UNTCNT                        0000020B  G
CDB_B_XMTCNT                        00000107  G
CDB_B_XMTMAP                        0000001B  G
CDB_C_ABORTS                  =     00000001
CDB_C_LENGTH                        000002F0  G
CDB_C_MAPPED                  =     000000A8
CDB_C_QUEUES                  =     00000005
CDB_C_SETPRM                  =     00000001
CDB_C_ZERO                          0000020A  G
CDB_G_COUNTER                       00000120  G
CDB_G_HWA                           00000250  G
CDB_G_MAPPED                        00000162  G
CDB_G_MLTTBL                        000002A6  G
CDB_G_MULTI                         0000025E  G
CDB_G_PHA                           0000024A  G
CDB_G_PHYADR                        00000256  G
CDB_G_RRING                         00000162  G
CDB_G_XRING                         000001CE  G
CDB_L_BIDCTR                        0000014A  G
CDB_L_BRCCTR                        0000012E  G
CDB_L_BS1CTR                        00000146  G
CDB_L_BSMCTR                        00000142  G
CDB_L_BSNCTR                        0000014E  G
CDB_L_CSR                           00000114  G
CDB_L_DBRCTR                        00000122  G
CDB_L_DBSCTR                        0000013A  G
CDB_L_DEVDEPEND                     0000010C  G
CDB_L_FPC                           0000000C  G
CDB_L_FQBL                          00000004  G
CDB_L_FQFL                          00000000  G
CDB_L_FR3                           00000010  G
CDB_L_FR4                           00000014  G
CDB_L_LASTCSR                       00000010  G
CDB_L_MBLCTR                        00000126  G
CDB_L_MBSCTR                        0000013E  G
CDB_L_MBYCTR                        00000132  G
CDB_L_MSNCTR                        00000152  G
CDB_L_PRMUSER                       00000210  G
CDB_L_RCVMAP                        0000001C  G
CDB_L_RCV_PA                        000000AC  G
CDB_L_RCV_VA                        000000C4  G
CDB_L_RINGMAP                       0000011C  G
CDB_L_RRINGPA                       00000044  G
CDB_L_RRINGVA                       0000007C  G
CDB_L_TQE                           00000214  G
CDB_L_UCBO                          00000110  G
CDB_L_UV1BUF                        0000020C  G
CDB_L_XMTMAP                        00000038  G
```

```
CDB_L_XMT_PA                    000000C0  G
CDB_L_XMT_VA                    000000D8  G
CDB_L_XRINGPA                   00000068  G
CDB_L_XRINGVA                   0000009C  G
CDB_MOD_M_MULTI               = 00000001  G
CDB_MOD_M_PROM                = 00000002  G
CDB_MOD_V_MULTI               = 00000000  G
CDB_MOD_V_PROM                = 00000001  G
CDB_Q_POST                      000000FC  G
CDB_Q_QUEUES                    000000DC  G
CDB_Q_RCVBUF                    000000EC  G
CDB_Q_RCVPND                    000000F4  G
CDB_Q_XMTPND                    000000E4  G
CDB_Q_XMTREQ                    000000DC  G
CDB_STS_M_ERR                 = 00000010  G
CDB_STS_M_FORK_PEND           = 00000004  G
CDB_STS_M_INITED              = 00000001  G
CDB_STS_M_RUN                 = 00000002  G
CDB_STS_M_SETUP               = 00000020  G
CDB_STS_M_TIMER               = 00000008  G
CDB_STS_V_ERR                 = 00000004  G
CDB_STS_V_FORK_PEND           = 00000002  G
CDB_STS_V_INITED              = 00000000  G
CDB_STS_V_RUN                 = 00000001  G
CDB_STS_V_SETUP               = 00000005  G
CDB_STS_V_TIMER               = 00000003  G
CDB_W_BSZ                       00000108  G
CDB_W_CDCCTR                    0000015A  G
CDB_W_DIAG2                     0000011A  G
CDB_W_LBECTR                    00000138  G
CDB_W_MODE                      00000244  G
CDB_W_OVRCTR                    00000136  G
CDB_W_QUOTA                     0000010A  G
CDB_W_RFLCTR                    0000012C  G
CDB_W_RFLMAP                    0000012A  G
CDB_W_SBUCTR                    0000015E  G
CDB_W_SFLCTR                    00000158  G
CDB_W_SFLMAP                    00000156  G
CDB_W_SIZE                      00000008  G
CDB_W_UBUCTR                    00000160  G
CDB_W_UFDCTR                    0000015C  G
CDB_W_ZERO                      00000120  G
CHANGE_PARAM                    00002684  R    03
CHECKER                         000023A7  R    03
CHECKPKT                        000023C1  R    03
CHECK_BUFS                      00000DBA  R    03
CHECK_P1                        00000DE3  R    03
CHECK_P2                        00000DBC  R    03
CHECK_PARAM                     00000B94  R    03
CHECK_QUOTA                     00000A35  R    03
CHECK_SHR                       00000735  R    03
CHECK_SPC                       000019B2  R    03
CHMODE                          00000EA6  R    03
CIRCUIT_PARAM                   000000FE  R    03
CIRC_CTR                        00000159  R    03
CIRC_CTR_BUFSIZ               = 00000020
CIRC_CTR_SIZE                 = 00000006
CLEANUP_SHR                     0000228B  RG   03
CLONED_UCB                      000001B7  RG   03
COM$DECATTNAST                  ********    X  03
COM$DRVDEALMEM                  ********    X  03
COM$FLUSHATTNS                  ********    X  03
COM$POST                        ********    X  03
COM$SETATTNAST                  ********    X  03
CONTROL_INIT                    000001B6  RG   03
COPY_RCV                        000019BF  R    03
CRB$C_AUXSTRUC                = 00000010
CRB$L_INTD                    = 00000024
CSR                             0000000E  G
CXB$B_CODE                    = 0000000B
CXB$B_R_FLAGS                   0000000B  G
CXB$B_TYPE                    = 0000000A
CXB$B_XQ_FUNC                   00000020  G
CXB$B_XQ_RING                   00000023  G
CXB$B_XQ_SLOT                   00000022  G
CXB$C_HEADER                  = 00000048
CXB$C_TRAILER                 = 00000004
CXB$G_R_DEST                    00000038  G
CXB$G_R_SRC                     0000003E  G
CXB$L_BC                      = 00000004
CXB$L_END_ACTION              = 0000001C
CXB$L_FL                      = 00000000
CXB$L_IRP                     = 00000014
CXB$L_LINK                    = 00000010
CXB$L_SPARE0                  = 00000024
CXB$L_SPARE1                  = 00000020
CXB$L_T_IRP                     00000024  G
CXB$L_T_UCB                     00000024  G
CXB$Q_STATION                 = 00000028
CXB$T_R_DATA                    00000038  G
CXB$T_R_USERDAT                 00000046  G
CXB$T_T_DATA                    0000003A  G
CXB$W_BCNT                    = 0000001A
CXB$W_BOFF                    = 00000018
CXB$W_LENGTH                  = 0000000C
CXB$W_R_NCHAIN                  0000001C  G
CXB$W_R_PTYPE                   00000044  G
CXB$W_R_SIZE                    00000046  G
CXB$W_R_STS                     00000014  G
CXB$W_SIZE                    = 00000008
CXB$W_XQ_RID                    00000022  G
CXB_CHECKER                     000023E4  R    03
CXB_CHECKPKT                    00002425  R    03
DC$_SCOM                      = 00000020
DDB$L_DDT                     = 0000000C
DDB$L_UCB                     = 00000004
DELETE_BLOCK                    0000226C  R    03
DELETE_SHR                      000022AD  RG   03
DEV$M_AVL                     = 00040000
DEV$M_IDV                     = 04000000
DEV$M_NET                     = 00002000
DEV$M_ODV                     = 08000000
DEV$M_SHR                     = 00010000
DEV_TIMEOUT                     00001E8B  RG   03
```

```
DIAG_B_SPARE                    0000000B
DIAG_B_TYPE                     0000000A
DIAG_C_EXTRA                =   00000006
DIAG_C_LENGTH                   0000003E
DIAG_G_DEST                     00000030
DIAG_G_HWA                      0000002A
DIAG_G_SRC                      00000036
DIAG_L_BUFFER                   00000004
DIAG_L_DATA                     00000000
DIAG_L_DEPEND                   00000024
DIAG_L_ERRS                     0000001C
DIAG_L_EXTRA                    00000020
DIAG_Q_FINISH                   00000014
DIAG_Q_START                    0000000C
DIAG_T_DATA                     0000000C
DIAG_T_RDATA                    00000030
DIAG_W_CSR                      00000024
DIAG_W_ERR                      00000026
DIAG_W_ERR2                     00000028
DIAG_W_SIZE                     00000008
DIAG_W_TYPE                     0000003C
DPT$B_FLAGS                 =   0000000D
DPT$C_LENGTH                =   00000038
DPT$C_VERSION               =   00000004
DPT$INITAB                      00000038 R      02
DPT$M_NOUNLOAD              =   00000004
DPT$REINITAB                    0000009F R      02
DPT$TAB                         00000000 R      02
DSC$A_POINTER               =   00000004
DTS_DEQNA                   =   00000016
DYN$C_BUFIO                 =   00000013
DYN$C_CDB                   =   00000033
DYN$C_CRB                   =   00000005
DYN$C_CXB                   =   0000001B
DYN$C_DDB                   =   00000006
DYN$C_DPT                   =   0000001E
DYN$C_IRP                   =   0000000A
DYN$C_ORB                   =   00000049
DYN$C_TQE                   =   0000000F
DYN$C_UCB                   =   00000010
EXE$ABORTIO                     ********    X   03
EXE$ALLOCBUF                    ********    X   03
EXE$ALONONPAGED                 ********    X   03
EXE$ALOPHYCNTG                  ********    X   03
EXE$BUFFRQUOTA                  ********    X   03
EXE$BUFQUOPRC                   ********    X   03
EXE$DEANONPAGED                 ********    X   03
EXE$FINISHIO                    ********    X   03
EXE$FORK                        ********    X   03
EXE$GB_CPUTYPE                  ********    X   03
EXE$GL_TENUSEC                  ********    X   03
EXE$GL_UBDELAY                  ********    X   03
EXE$GQ_SYSTIME                  ********    X   03
EXE$INSTIMQ                     ********    X   03
EXE$PROBER_DSC                  ********    X   03
EXE$QIORETURN                   ********    X   03
EXE$READCHK                     ********    X   03
EXE$WRITECHK                    ********    X   03
FFI$L_DL_UCB                =   00000034
FFI$L_ERROR                 =   0000001C
FFI$L_RECV_DONE             =   00000018
FFI$L_SHUT_DONE             =   00000020
FFI$L_XMIT                  =   00000010
FFI$L_XMIT_DONE             =   00000014
FFI_INIT                        0000026A RG     03
FILERCVLIST                     00001433 RG     03
FIND_MLTENTRY                   00002926 R      03
FIND_SHR                        00002457 R      03
FINISH_RCV_FFI                  00001A4F RG     03
FINISH_RCV_IO                   00001A8A RG     03
FINISH_XMT_FFI                  00001A41 RG     03
FKB$B_FIPL                  =   0000000B
FKB$B_TYPE                  =   0000000A
FKB$L_FR3                   =   00000010
FORK_PROC                       00001663 RG     03
FORK_TIMER                      00001321 R      03
FUNCTAB_LEN                 =   00000040
GET_CHAR_BUF                    00000D7D R      03
IDB$B_VECTOR                =   0000000B
IDB$L_CSR                   =   00000000
IDB$L_UCBLST                =   00000018
INACT_ERROR                     000006C4 R      03
INIT_C_AQUOTA               =   00000002
INIT_C_BUFSIZE              =   00000080
INIT_C_QUOTA                =   00002328
INTEXIT                         0000163A R      03
IO$V_ATTNAST                =   00000008
IO$V_CLR_COUNT              =   0000000A
IO$V_CTRL                   =   00000009
IO$V_NOW                    =   00000006
IO$V_RD_COUNT               =   00000008
IO$V_SHUTDOWN               =   00000007
IO$V_STARTUP                =   00000006
IO$_READLBLK                =   00000021
IO$_READPBLK                =   0000000C
IO$_READVBLK                =   00000031
IO$_SENSECHAR               =   0000001B
IO$_SENSEMODE               =   00000027
IO$_SETCHAR                 =   0000001A
IO$_SETMODE                 =   00000C23
IO$_VIRTUAL                 =   0000003F
IO$_WRITELBLK               =   00000020
IO$_WRITEPBLK               =   0000000B
IO$_WRITEVBLK               =   00000030
IOC$ALOUBAMAP                   ********    X   03
IOC$CREDIT_UCB                  ********    X   03
IOC$DELETE_UCB                  ********    X   03
IOC$INITIATE                    ********    X   03
IOC$LOADUBAMAP                  ********    X   03
IOC$LOADUBAMAPA                 ********    X   03
IOC$LOADUBAMAPN                 ********    X   03
IOC$MNTVER                      ********    X   03
IOC$PURGDATAP                   ********    X   03
IOC$RELDATAP                    ********    X   03
```

```
IOC$RELMAPREG                   ********   X    03
IOC$RETURN                      ********   X    03
IO_DONE                         00001B50 R      03
IO_DONE1                        00001B54 R      03
IO_DONE2                        00001B6A R      03
IO_DONE3                        00001B3C R      03
IPL$_ASTDEL                   = 00000002
IPL$_QUEUEAST                 = 00000006
IPL$_SYNCH                    = 00000008
IPL$_TIMER                    = 00000008
IPL$_XQ_DIPL                  = 00000015
IPL$_XQ_FIPL                  = 00000008
IRP$B_EFN                     = 00000022
IRP$B_PRI                     = 00000023
IRP$B_RMOD                    = 0000000B
IRP$B_TYPE                    = 0000000A
IRP$B_XQ_DATAP                  00000046  G
IRP$B_XQ_FUNC                   00000021  G
IRP$B_XQ_RING                   00000039  G
IRP$B_XQ_SLOT                   00000038  G
IRP$C_LENGTH                  = 000000C4
IRP$C_XQ_STD                    00000060  G
IRP$L_AST                     = 00000010
IRP$L_ASTPRM                  = 00000014
IRP$L_BCNT                    = 00000032
IRP$L_DIAGBUF                 = 0000004C
IRP$L_FQFL                    = 00000060
IRP$L_IOSB                    = 00000024
IRP$L_IOST1                   = 00000038
IRP$L_IOST2                   = 0000003C
IRP$L_LBOFF                   = 00000090
IRP$L_MEDIA                   = 00000038
IRP$L_PID                     = 0000000C
IRP$L_RBOFF                   = 00000098
IRP$L_RBUFH_AD                = 00000094
IRP$L_SVAPTE                  = 0000002C
IRP$L_UCB                     = 0000001C
IRP$L_WIND                    = 00000018
IRP$L_XQ_DATBUF                 0000003C  G
IRP$L_XQ_DGUNI                  00000040  G
IRP$L_XQ_MAP                    0000003C  G
IRP$L_XQ_P2BUF                  00000040  G
IRP$L_XQ_SETUP                  00000094  G
IRP$L_XQ_SHR                    00000090  G
IRP$L_XQ_SYSBUF                 0000003C  G
IRP$L_XQ_UPADR                  00000044  G
IRP$L_XQ_USERBUF                0000003C  G
IRP$M_CHAINED                 = 00000020
IRP$M_COMPLX                  = 00000008
IRP$M_DIAGBUF                 = 00000080
IRP$Q_STATION                 = 00000040
IRP$V_CHAINED                 = 00000005
IRP$V_DIAGBUF                 = 00000007
IRP$V_FUNC                    = 00000001
IRP$W_BCNT                    = 00000032
IRP$W_BOFF                    = 00000030
IRP$W_CHAN                    = 00000028
IRP$W_FUNC                    = 00000020
IRP$W_SIZE                    = 00000008
IRP$W_STS                     = 0000002A
IRP$W_XQ_CODE                   00000040  G
IRP$W_XQ_P2SIZ                  00000044  G
IRP$W_XQ_PROTYP                 0000003A  G
IRP$W_XQ_RID                    00000038  G
IRP$W_XQ_STATUS                 0000003A  G
IRP$W_XQ_USERSIZ                00000038  G
JIB$L_BYTCNT                  = 00000020
JIB$L_BYTLM                   = 00000024
LINE_CTR                        00000109 R      03
LINE_CTR_BUFSIZ               = 0000006A
LINE_CTR_SIZE                 = 00000014
LINE_PARAM                      00000081 R      03
LINE_PARAM_WO                   00000078 R      03
LINE_PRM_BUFSIZ               = 0000007A
LOAD_PECO                       000015D8 RG     03
MASKA                         = 00000080
MASKL                         = 08000000
MATCH_ADDRESS                   00002A3E R      03
MATCH_MULTI                     00002A0F R      03
MATCH_PROMTYP                   00002ABD R      03
MATCH_PROTYP                    00002A9B R      03
MATCH_SHR                       00000710 R      03
MATCH_SRC                       0000198B R      03
MAX_BUFSIZ_UV1                = 000005EA
MAX_C_CHAIN                   = 00000001
MAX_C_MLT                     = 0000000C
MAX_C_RCV                     = 00000008
MAX_C_RCVUV1                  = 00000005
MAX_C_XMT                     = 00000004
MAX_C_XMTUV1                  = 00000001
MAX_PKT_SIZE                  = 000005DC
MIN_PKT_SIZE                  = 0000002E
MMG$GL_SPTBASE                  ********   X    03
MOPCTRTAB                       00000171 R      03
MOP_CTR_BUILD                   00001D36 R      03
MOP_CTR_REQUEST                 00001D01 RG     03
MOP_CTR_SIZE                  = 00000051
MOVE_MULTI                      00002507 R      03
NEXTMSG                         00001C39 R      03
NI_CTR_PROTYP                 = 00000260
NI_CTR_READ                   = 00000009
NI_CTR_REPLY                  = 00000008
NMA$C_ACC_EXC                 = 00000003
NMA$C_ACC_LIM                 = 00000002
NMA$C_ACC_SHR                 = 00000001
NMA$C_CTCIR_BRC               = 000003E8
NMA$C_CTCIR_BSN               = 000003E9
NMA$C_CTCIR_DBR               = 000003F2
NMA$C_CTCIR_DBS               = 000003F3
NMA$C_CTCIR_MNE               = 00000A8D
NMA$C_CTCIR_UBU               = 00000429
NMA$C_CTLIN_BID               = 000003F5
NMA$C_CTLIN_BRC               = 000003E8
NMA$C_CTLIN_BS1               = 000003F6
```

```
NMA$C_CTLIN_BSM         = 000003F7
NMA$C_CTLIN_BSN         = 000003E9
NMA$C_CTLIN_CDC         = 00000425
NMA$C_CTLIN_DBR         = 000003F2
NMA$C_CTLIN_DBS         = 000003F3
NMA$C_CTLIN_LBE         = 00000411
NMA$C_CTLIN_MBL         = 000003F4
NMA$C_CTLIN_MBS         = 000C0A8D
NMA$C_CTLIN_MBY         = 000003EA
NMA$C_CTLIN_MSN         = 00000A8E
NMA$C_CTLIN_OVR         = 00000428
NMA$C_CTLIN_RFL         = 00000426
NMA$C_CTLIN_SBU         = 00000429
NMA$C_CTLIN_SFL         = 00000424
NMA$C_CTLIN_UBU         = 0000042A
NMA$C_CTLIN_UFD         = 00000427
NMA$C_CTLIN_ZER         = 00000000
NMA$C_LINCN_LOO         = 00000001
NMA$C_LINCN_NOR         = 00000000
NMA$C_LINMC_CAL         = 00000003
NMA$C_LINMC_CLR         = 00000002
NMA$C_LINMC_SET         = 00000004
NMA$C_LINMC_S_T         = 00000001
NMA$C_LINPR_NI          = 00000006
NMA$C_LINPR_POI         = 00000000
NMA$C_PCCI_MST          = 00000AFA
NMA$C_PCLI_ACC          = 00000B1E
NMA$C_PCLI_BFN          = 00000451
NMA$C_PCLI_BSZ          = 00000820
NMA$C_PCLI_BUS          = 00000AF1
NMA$C_PCLI_CON          = 00000456
NMA$C_PCLI_CRC          = 00000B1C
NMA$C_PCLI_DCH          = 00000B1B
NMA$C_PCLI_DES          = 00000821
NMA$C_PCLI_HBQ          = 00000B1D
NMA$C_PCLI_HWA          = 00000488
NMA$C_PCLI_MCA          = 00000B0F
NMA$C_PCLI_MLT          = 00000819
NMA$C_PCLI_PAD          = 00000B1A
NMA$C_PCLI_PHA          = 00000804
NMA$C_PCLI_PRM          = 00000818
NMA$C_PCLI_PRO          = 00000458
NMA$C_PCLI_PTY          = 00000B0E
NMA$C_STATE_OFF         = 00000001
NMA$C_STATE_ON          = 00000000
NMA$M_CNT_COU           = 00008000
NMA$M_CNT_MAP           = 00001000
NMA$M_CNT_TYP           = 00000FFF
NMA$V_CNT_MAP           = 0000000C
NMA$V_CNT_WID           = 0000000D
NO_SHR                    00000450 R      03
ORB$B_FLAGS             = 0000000B
ORB$L_OWNER             = 00000000
ORB$M_PROT_16           = 00000001
ORB$W_PROT              = 00000018
P1                      = 00000000
P2                      = 00000004
P2B_B_SPARE               0000000B
P2B_B_TYPE                0000000A
P2B_C_LENGTH              0000000C
P2B_L_BUFFER              00000004
P2B_L_POINTER             00000000
P2B_T_DATA                0000000C
P2B_W_SIZE                00000008
P3                      = 00000008
P4                      = 0000000C
P5                      = 00000010
PCB$L_JIB               = 00000080
PCB$L_PID               = 00000060
PCB$L_STS               = 00000024
PCB$Q_PRIV              = 00000084
PCB$V_SSRWAIT           = 0000000A
PHYADD0                   00000000  G
PHYADD1                   00000002  G
PHYADD2                   00000004  G
PHYADD3                   00000006  G
PHYADD4                   00000008  G
PHYADD5                   0000000A  G
POKE_USER                 00002A66 R      03
PR$_IPL                 = 00000012
PR$_SID_TYP730          = 00000003
PR$_SID_TYP750          = 00000002
PR$_SID_TYP780          = 00000001
PR$_SID_TYP790          = 00000004
PR$_SID_TYPUV1          = 00000007
PR$_SID_TYPUV2          = 00000008
PRM_B_FLAG                00000002
PRM_FLG_M_CHECK         = 00000010
PRM_FLG_M_INVALID       = 00000004
PRM_FLG_M_MAX           = 00000002
PRM_FLG_M_MIN           = 00000001
PRM_FLG_V_CDB           = 00000003
PRM_FLG_V_INVALID       = 00000002
PRM_FLG_V_MAX           = 00000001
PRM_FLG_V_MIN           = 00000000
PRM_OFF_M_VALUE         = 000003FF
PRM_OFF_M_WIDTH         = 0000FC00
PRM_OFF_S_VALUE         = 0000000A
PRM_OFF_S_WIDTH         = 00000006
PRM_OFF_V_VALUE         = 00000000
PRM_OFF_V_WIDTH         = 0000000A
PRM_TYP_M_CODE          = 00000FFF
PRM_TYP_M_STRING        = 00001000
PRM_TYP_V_STRING        = 0000000C
PRM_W_OFF                 00000003
PRM_W_TYPE                00000000
PRV$V_PHY_IO            = 00000016
PTE$S_PFN               = 00000015
QNA_IMTR                  000015FF RG     03
QUEPKT                    00000CB3 R      03
RCVLIST                   00000004  G
RCVLST1                   00000006  G
RCV_C_LENGTH              0000000C  G
RCV_DSC_M_CHAIN         = 00004000  G
```

```
RCV_DSC_M_VALID          =  00008000   G
RCV_DSC_S_CHAIN          =  00000001   G
RCV_DSC_S_VALID          =  00000001   G
RCV_DSC_V_CHAIN          =  0000000E   G
RCV_DSC_V_VALID          =  0000000F   G
RCV_ERROR                   00001910 R       03
RCV_FDT                     00000659 RG      03
RCV_FLG_M_ERR            =  00004000   G
RCV_FLG_M_LAST           =  00008000   G
RCV_FLG_V_ERR            =  0000000E   G
RCV_FLG_V_LAST           =  0000000F   G
RCV_K_LENGTH             =  0000006C
RCV_START                   000006CA RG      03
RCV_STS_M_CRCERR         =  00000002   G
RCV_STS_M_DISCARD        =  00001000   G
RCV_STS_M_ERR            =  00004000   G
RCV_STS_M_ESETUP         =  00002000   G
RCV_STS_M_FRAME          =  00000004   G
RCV_STS_M_LAST           =  00008000   G
RCV_STS_M_OVF            =  00000001   G
RCV_STS_M_RLEN           =  00000700   G
RCV_STS_M_RUNT           =  00000800   G
RCV_STS_M_SHORT          =  00000008   G
RCV_STS_S_RLEN           =  00000003   G
RCV_STS_V_CRCERR         =  00000001   G
RCV_STS_V_DISCARD        =  0000000C   G
RCV_STS_V_ERR            =  0000000E   G
RCV_STS_V_ESETUP         =  0000000D   G
RCV_STS_V_FRAME          =  00000002   G
RCV_STS_V_LAST           =  0000000F   G
RCV_STS_V_OVF            =  00000000   G
RCV_STS_V_RLEN           =  00000008   G
RCV_STS_V_RUNT           =  0000000B   G
RCV_STS_V_SHORT          =  00000003   G
RCV_W_ADDR                  00000004   G
RCV_W_ADDRHI                00000002   G
RCV_W_FLAG                  00000000   G
RCV_W_LEN                   00000006   G
RCV_W_LENB                  0000000A   G
RCV_W_STS                   00000008   G
READ_CIRC_CTR               00000CE9 R       03
READ_LINE_CTR               00000CCA R       03
REG_DUMP                    00001D80 RG      03
RESTART_DELTA            =  01C9C380
RESTART_ROUT                00001DCC RG      03
RES_MULTI                   00002533 R       03
RETURN_IRP                  00001E25 R       03
RETURN_MULTI                000029DB R       03
RETURN_P2                   00002771 RG      03
RHDR_B_SPARE                00000008
RHDR_B_TYPE                 0000000A
RHDR_C_DATA              =  0000000E
RHDR_C_LENGTH               0000001A
RHDR_G_DEST                 0000000C
RHDR_G_SRC                  00000012
RHDR_L_BUFFER               00000004
RHDR_L_DATA                 00000000
RHDR_T_DATA                 0000000C
RHDR_W_SIZE                 00000008
RHDR_W_TYPE                 00000018
SAV_MULTI                   00002510 R       03
SCH$GL_PCBVEC               ******** RX      03
SCH$IOLOCKW                 ******** RX      03
SCH$IOUNLOCK                ******** RX      03
SCHED_FORK                  00001646 RG      03
SCHED_FORKC                 00001644 RG      03
SCS$GB_SYSTEMID             ******** RX      03
SENSEMODE_FDT               000008FE RG      03
SETMODE_FDT                 0000079E RG      03
SETUP_ERR                   0000135E R       03
SETUP_MODE                  00001364 R       03
SET_DESAD                   00002988 R       03
SET_MULTI                   00002880 R       03
SET_MULTIN                  000008E0 R       03
SET_PHYAD                   0000297D R       03
SHR_B_STS                   00000008   G
SHR_B_TYPE                  0000000A   G
SHR_C_LENGTH                0000002A   G
SHR_C_QUEUES             =  00000002
SHR_DEF                     0000042F R       03
SHR_G_DEST                  00000012   G
SHR_L_PID                   0000000C   G
SHR_L_QBL                   00000004   G
SHR_L_QFL                   00000000   G
SHR_Q_QUEUES                00000018   G
SHR_Q_RCVMSG                00000018   G
SHR_Q_RCVREQ                00000020   G
SHR_STS_M_INITED         =  00000001   G
SHR_STS_V_INITED         =  00000000   G
SHR_UCB                     00000A6E RG      03
SHR_W_CHAN                  00000010   G
SHR_W_QUOTA                 00000028   G
SHR_W_SIZE                  00000008   G
SHUT                        00000E7A R       03
SHUTDOWN                    0000203D RG      03
SHUTDOWN_PROTYP             0000201E RG      03
SHUTDOWN_QNA                00001F0D RG      03
SIZ...                   =  00000006
SQUEEZ_MULTI                0000294A R       03
SS$_ABORT                =  0000002C
SS$_ACCVIO               =  0000000C
SS$_BADPARAM             =  00000014
SS$_BUFFEROVF            =  00000601
SS$_COMMHARD             =  000020C4
SS$_CTRLERR              =  00000054
SS$_DATACHECK            =  0000005C
SS$_DATAOVERUN           =  00000838
SS$_DEVALLOC             =  00000840
SS$_DEVINACT             =  000020D4
SS$_DEVOFFLINE           =  00000084
SS$_DEVREQERR            =  00000334
SS$_DISCONNECT           =  0000204C
SS$_DUPUNIT              -  000021C4
SS$_ENDOFFILE            =  00000870
```

```
SS$_EXQUOTA                =  0000001C
SS$_INSFMAPREG             =  00000344
SS$_INSFMEM                =  00000124
SS$_IVBUFLEN               =  0000034C
SS$_NOPRIV                 =  00000024
SS$_NORMAL                 =  00000001
SS$_OPINCOMPL              =  000002D4
SS$_POWERFAIL              =  00000364
SS$_TIMEOUT                =  0000022C
START                       00000EB5 RG     03
STARTIO                     00000E4B RG     03
STARTUP                     00000E6A R      03
ST  T_RECEIVE               000014AF RG     03
SIART_TIMER                 00001332 R      03
STOP                        00000E9E R      03
STOP_TQE                    00001E68 R      03
TAKE_QUOTA                  000009DC R      03
TIMEOUT                     00001E7A RG     03
TIMOUT                      00001E5D R      03
TOSAMSG                     00001C10 R      03
TOSSMSG                     00001C13 R      03
TQE$B_RQTYPE               =  00000008
TQE$B_TYPE                 =  0000000A
TQE$C_LENGTH               =  00000030
TQE$C_SSREPT               =  00000005
TQE$C_SSSNGL               =  00000001
TQE$L_FPC                  =  0000000C
TQE$L_RQPID                =  0000002C
TQE$M_REPEAT               =  00000004
TQE$Q_DELTA                =  00000020
TQE$W_SIZE                 =  00000008
TQE_C_DELTA                =  00000002
TQE_DELTA                  =  01312D00
TQE_TIMER                   00001E2F RG     03
UCB$B_DEVCLASS             =  00000040
UCB$B_DEVTYPE              =  00000041
UCB$B_DIPL                 =  0000005E
UCB$B_FIPL                 =  00000008
UCB$B_XQ_ACC                000000D8  G
UCB$B_XQ_BFN                000000D9  G
UCB$B_XQ_CDBPRM             000000E1  G
UCB$B_XQ_CON                000000E1  G
UCB$B_XQ_DCH                000000E0  G
UCB$B_XQ_MLT                000000DF  G
UCB$B_XQ_MLTTBL             000000EA  G
UCB$B_XQ_MST                000000E8  G
UCB$B_XQ_MULTI              000000E9  G
UCB$B_XQ_PAD                000000DD  G
UCB$B_XQ_PRM                000000DE  G
UCB$B_XQ_PRO                000000DC  G
UCB$B_XQ_SETPRM             000000D0  G
UCB$B_XQ_SHRPRM             000000DA  G
UCB$B_XQ_SPARE              000000EB  G
UCB$C_NI_LENGTH            =  00000098
UCB$C_XQ_CDBPRM            =  00000001
UCB$C_XQ_LENGTH             0000019A  G
UCB$C_XQ_QUEUES            =  00000004
UCB$C_XQ_SETPRM            =  00000007
UCB$C_XQ_SHRPRM            =  00000008
UCB$G_XQ_DES                000000D0  G
UCB$G_XQ_MLTTBL             00000134  G
UCB$G_XQ_MULTI              000000EC  G
UCB$G_XQ_PHA                000000E2  G
UCB$L_CPID                 =  00000020
UCB$L_CRB                  =  00000024
UCB$L_DDB                  =  00000028
UCB$L_DEVCHAR              =  00000038
UCB$L_DEVDEPEND            =  00000044
UCB$L_LINK                 =  00000030
UCB$L_NI_HWAPTR            =  00000090
UCB$L_NI_MLTPTR            =  00000094
UCB$L_ORB                  =  0000001C
UCB$L_PID                  =  0000002C
UCB$L_SVAPTE               =  00000078
UCB$L_XQ_AST                000000C0  G
UCB$L_XQ_CDB                000000C8  G
UCB$L_XQ_CPID               000000BC  G
UCB$L_XQ_DEFUSR             000000C4  G
UCB$L_XQ_FFI                00000192  G
UCB$L_XQ_PID                00000088  G
UCB$L_XQ_RBLCTR             00000188  G
UCB$L_XQ_RBYCTR             0000018C  G
UCB$L_XQ_SBLCTR             00000180  G
UCB$L_XQ_SBYCTR             00000184  G
UCB$L_XQ_STIRP              00000196  G
UCB$M_INT                  =  00000002
UCB$M_ONLINE               =  00000010
UCB$M_POWER                =  00000020
UCB$M_TEMPLATE             =  00002000
UCB$M_TIM                  =  00000001
UCB$M_XQ_INITED            =  00000001  G
UCB$M_XQ_INTERLOCK         =  00004000  G
UCB$M_XQ_PROTYP            =  00000004  G
UCB$M_XQ_RESTART           =  00008000  G
UCB$M_XQ_RUN               =  00000010  G
UCB$M_XQ_SHARE             =  00000008  G
UCB$M_XQ_STACK             =  00000040  G
UCB$M_XQ_START             =  00000020  G
UCB$Q_XQ_IOQS                000000A0  G
UCB$Q_XQ_QUEUES              00000098  G
UCB$Q_XQ_RCVMSG              000000A0  G
UCB$Q_XQ_RCVREQ              000000A8  G
UCB$Q_XQ_SHARE               00000098  G
UCB$Q_XQ_XMTREQ              00000080  G
UCB$V_ONLINE               =  00000004
UCB$V_POWER                =  00000005
UCB$V_XQ_INITED            =  00000000  G
UCB$V_XQ_INTERLOCK         =  0000000E  G
UCB$V_XQ_PROTYP            =  00000002  G
UCB$V_XQ_RESTART           =  0000000F  G
UCB$V_XQ_RUN               =  00000004  G
UCB$V_XQ_SHARE             =  00000003  G
UCB$V_XQ_STACK             =  00000006  G
UCB$V_XQ_START             =  00000005  G
```

```
UCB$W_BCNT              =  0000007E
UCB$W_BOFF              =  0000007C
UCB$W_DEVBUFSIZ         =  00000042
UCB$W_DEVSTS            =  00000068
UCB$W_ERRCNT            =  00000082
UCB$W_REFC              =  0000005C
UCB$W_STS               =  00000064
UCB$W_UNIT              =  00000054
UCB$W_XQ_BSZ               000000DA  G
UCB$W_XQ_CTR               0000017C  G
UCB$W_XQ_HBQ               000000D6  G
UCB$W_XQ_MNECTR            0000017C  G
UCB$W_XQ_PROTYP            000000CE  G
UCB$W_XQ_QUOTA             000000CC  G
UCB$W_XQ_TOTQUO            00000190  G
UCB$W_XQ_UBUCTR            0000017E  G
UNIT_INIT                  000001BA  RG     03
UV1_BUFFER_AREA         =  0000237C
UV1_BUFFER_LENGTH       =  00002400
UV1_BUFFER_PAGES        =  00000012
VA$M_BYTE               =  000001FF
VA$S_VPN                =  00000015
VA$V_VPN                =  00000009
VALIDATE_P2                00002549  RG     03
VALID_MULTI                0000283C  R      03
VALID_PHYAD                00002878  R      03
VEC$B_DATAPATH          =  00000013
VEC$B_NUMREG            =  00000012
VEC$L_IDB               =  00000008
VEC$L_INITIAL           =  0000000C
VEC$L_START             =  0000001C
VEC$L_UNITINIT          =  00000018
VEC$W_MAPREG            =  00000010
VECTOR                     0000000C  G
XBUF_C_HEADER              0000000E
XBUF_G_DEST                00000000
XBUF_G_SRC                 00000006
XBUF_T_DATA                0000000E
XBUF_W_SIZE                0000000E
XBUF_W_TYPE                00000000
XMSM_STS_ACTIVE         =  00000800
XMSV_ERR_FATAL          =  00000010
XMSV_ERR_START          =  00000017
XMSV_STS_ACTIVE         =  0000000B
XMSV_STS_BUFFAIL        =  00000000
XMSV_STS_TIMO           =  00000009
XMTLIST                    00000008  G
XMTLST1                    0000000A  G
XMT_ALT_START              00000487  RG     03
XMT_C_LENGTH               0000000C  G
XMT_C_TIM               =  00000006
XMT_DSC_M_BEGODD        =  00000040  G
XMT_DSC_M_CHAIN         =  00004000  G
XMT_DSC_M_ENDODD        =  00000080  G
XMT_DSC_M_EOM           =  00002000  G
XMT_DSC_M_SETUP         =  00001000  G
XMT_DSC_M_VALID         =  00008000  G
XMT_DSC_S_BEGODD        =  00000001  G
XMT_DSC_S_CHAIN         =  00000001  G
XMT_DSC_S_ENDODD        =  00000001  G
XMT_DSC_S_EOM           =  00000001  G
XMT_DSC_S_SETUP         =  00000001  G
XMT_DSC_S_VALID         =  00000001  G
XMT_DSC_V_BEGODD        =  00000006  G
XMT_DSC_V_CHAIN         =  0000000E  G
XMT_DSC_V_ENDODD        =  00000007  G
XMT_DSC_V_EOM           =  0000000D  G
XMT_DSC_V_SETUP         =  0000000C  G
XMT_DSC_V_VALID         =  0000000F  G
XMT_ERROR                  0000193C  R      03
XMT_FDT                    0000028B  RG     03
XMT_FFI_START              00000346  RG     03
XMT_FLG_M_ERR           =  00004000  G
XMT_FLG_M_LAST          =  00008000  G
XMT_FLG_V_ERR           =  0000000E  G
XMT_FLG_V_LAST          =  0000000F  G
XMT_INITIATE               000003CE  RG     03
XMT_K_LENGTH            =  0000003C
XMT_START                  00000389  RG     03
XMT_STS_M_COL           =  000000F0  G
XMT_STS_M_ERR           =  00004000  G
XMT_STS_M_LAST          =  00008000  G
XMT_STS_M_LCAR          =  00001000  G
XMT_STS_M_NOCAR         =  00000800  G
XMT_STS_S_COL           =  00000004  G
XMT_STS_V_ABORT         =  00000009  G
XMT_STS_V_COL           =  00000004  G
XMT_STS_V_ERR           =  0000000E  G
XMT_STS_V_FAIL          =  00000008  G
XMT_STS_V_LAST          =  0000000F  G
XMT_STS_V_LCAR          =  0000000C  G
XMT_STS_V_NOCAR         =  0000000B  G
XMT_TDR_M_TDR           =  00003FFF  G
XMT_TDR_S_TDR           =  0000000E  G
XMT_TDR_V_TDR           =  00000000  G
XMT_TIM                 =  00000004
XMT_UV1                    000005CD  R      03
XMT_W_ADDR                 00000004  G
XMT_W_ADDRHI               00000002  G
XMT_W_FLAG                 00000000  G
XMT_W_LEN                  00000006  G
XMT_W_STS                  00000008  G
XMT_W_TDR                  0000000A  G
XQ$DDT                     00000000  RG     03
XQ_CSR_M_CAR            =  00002000  G
XQ_CSR_M_ELOOP          =  00000200  G
XQ_CSR_M_ERR            =  00008000  G
XQ_CSR_M_ILOOP          =  00000100  G
XQ_CSR_M_INTENA         =  00000040  G
XQ_CSR_M_NXM            =  0C000004  G
XQ_CSR_M_RCVENA         =  00000001  G
XQ_CSR_M_RCVINT         =  00000080  G
XQ_CSR_M_RCVINV         =  00000020  G
XQ_CSR_M_RESET          =  00000002  G
```

```
XQ_CSR_M_RROM                   = 00000008  G
XQ_CSR_M_SANITY                 = 00000400  G
XQ_CSR_M_XCAB                   = 00001000  G
XQ_CSR_M_XMTINT                 = 00008000  G
XQ_CSR_M_XMTINV                 = 00000010  G
XQ_CSR_V_CAR                    = 0000000D  G
XQ_CSR_V_ELOOP                  = 00000009  G
XQ_CSR_V_ERR                    = 0000000E  G
XQ_CSR_V_ILOOP                  = 00000008  G
XQ_CSR_V_INTENA                 = 00000006  G
XQ_CSR_V_NXM                    = 00000002  G
XQ_CSR_V_RCVENA                 = 00000000  G
XQ_CSR_V_RCVINT                 = 00000007  G
XQ_CSR_V_RCVINV                 = 00000005  G
XQ_CSR_V_RESET                  = 00000001  G
XQ_CSR_V_RROM                   = 00000003  G
XQ_CSR_V_SANITY                 = 0000000A  G
XQ_CSR_V_XCAB                   = 0000000C  G
XQ_CSR_V_XMTINT                 = 0000000F  G
XQ_CSR_V_XMTINV                 = 00000004  G
XQ_C_ADDRCV                     = 00000040
XQ_C_CNTSIZ                     = 00000002
XQ_C_CRC                        = 00000004
XQ_C_HEADER                     = 0000000E
XQ_C_STPRO                      = 00000660
XQ_END                            00002ACA RG    03
XQ_FC_V_CANCEL                  = 00000004  G
XQ_FC_V_CHMODE                  = 00000006  G
XQ_FC_V_INIT                    = 00000000  G
XQ_FC_V_RECV                    = 00000002  G
XQ_FC_V_RESTART                 = 00000005  G
XQ_FC_V_STOP                    = 00000003  G
XQ_FC_V_XMIT                    = 00000001  G
XQ_FUNCTABLE                      00000038 R    03
XQ_SOFT_M_POWER                 = 00000002  G
XQ_SOFT_M_TIMEOUT               = 00000001  G
XQ_SOFT_V_POWER                 = 00000001  G
XQ_SOFT_V_TIMEOUT               = 00000000  G
_$$_                            = 00000000
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	000002F0 (752.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
$$$105_PROLOGUE	000000B9 (185.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE
$$$115_DRIVER	00002ACA (10954.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	100	00:00:00.03	00:00:01.80
Command processing	124	00:00:00.25	00:00:02.79
Pass 1	1893	00:00:28.61	00:01:59.27
Symbol table sort	0	00:00:01.95	00:00:07.43
Pass 2	480	00:00:09.16	00:01:07.86
Symbol table output	2	00:00:00.19	00.00:00.90
Psect synopsis output	2	00:00:00.01	00:00:00.01
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	2604	00:00:40.20	00:03:20.06

The working set limit was 3900 pages.
548759 bytes (1072 pages) of virtual memory were used to buffer the intermediate code.
There were 250 pages of symbol table space allocated to hold 4247 non-local and 632 local symbols.
8278 source lines were read in Pass 1, producing 64 object records in Pass 2.
103 pages of virtual memory were used to define 92 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA18:[SHRLIB]NMALIBRY.MLB;1	1
_255DUA18:[SYS.OBJ]LIB.MLB;1	42
_255DUA18:[SYSLIB]STARLET.MLB;3	14
TOTALS (all libraries)	57

4109 GETS were required to define 57 macros.

There were no errors, warnings or information messages.

```
MACRO/LIS=LIS$:XQDRIVER/OBJ=OBJ$:XQDRIVER MSRC$:XQDRIVER/UPDATE=(BUG$:XQDRIVER)+EXECML$/LIB+SHRLIB$:NMALIBRY/LIB
```

0440 AH-EF71A-SE
VAX/VMS V4.1 SRC LST MCRF UPD

XQDRIVER
MAP

XQDRIVER
LIS

TSDRIVER
LIS

0441 AH-EF71A-SE
VAX/VMS V4.1 SRC LST MCRF UPD

VAXARITH
LIS

EMULAT

ERFPROC1
MAP

VAXEMUL
MAP

ERF

ERF
MAP

REGSELECT
LIS