

```
DDDDDDDDDDDD  RRRRRRRRRR  IIIIIIIII  VVV  VVV  EEEEEEEEEEEEE  RRRRRRRRRR
DDDDDDDDDDDD  RRRRRRRRRR  IIIIIIIII  VVV  VVV  EEEEEEEEEEEEE  RRRRRRRRRR
DDDDDDDDDDDD  RRRRRRRRRR  IIIIIIIII  VVV  VVV  EEEEEEEEEEEEE  RRRRRRRRRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDD  DDD  RRR  RRR  RRR  VVV  VVV  EEE  RRR  RRR
DDDDDDDDDDDD  RRR  RRR  IIIIIIIII  VVV  VVV  EEEEEEEEEEEEE  RRR  RRR
DDDDDDDDDDDD  RRR  RRR  IIIIIIIII  VVV  VVV  EEEEEEEEEEEEE  RRR  RRR
DDDDDDDDDDDD  RRR  RRR  IIIIIIIII  VVV  VVV  EEEEEEEEEEEEE  RRR  RRR
```

```

XX      XX      QQQQQQ      DDDDDDDD      RRRRRRRR      IIIIII      VV      VV      EEEEEEEEE      RRRRRRRR
XX      XX      QQQQQQ      DDDDDDDD      RRRRRRRR      IIIIII      VV      VV      EEEEEEEEE      RRRRRRRR
XX      XX      QQ      QQ      DD      DD      RR      RR      EE      RR      RR
XX      XX      QQ      QQ      DD      DD      RR      RR      EE      RR      RR
  XX  XX      QQ      QQ      DD      DD      RR      RR      EE      RR      RR
  XX  XX      QQ      QQ      DD      DD      RRRRRRRR      II      EE      RRRRRRRR
    XX  XX      QQ      QQ      DD      DD      RRRRRRRR      II      EEEEEEEE      RRRRRRRR
    XX  XX      QQ      QQ      DD      DD      RR      RR      EE      RR      RR
  XX  XX      QQ      QQ      DD      DD      RR      RR      EE      RR      RR
XX      XX      QQ      QQ      DD      DD      RR      RR      EE      RR      RR
XX      XX      QQ      QQ      DD      DD      RR      RR      EE      RR      RR
XX      XX      QQQQ      QQ      DDDDDDDD      IIIIII      VV      VV      EEEEEEEEE      RR      RR
XX      XX      QQQQ      QQ      DDDDDDDD      IIIIII      VV      VV      EEEEEEEEE      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
LLLLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLLLL      IIIIII      SSSSSSSS

```

XQ
VC

:R

:R

(9)	1121	Standard tables
(10)	1204	Local driver storage
(11)	1439	CONTROL_INIT - INITIALIZE DEQNA DEVICE
(12)	1465	CLONED_UCB - INITIALIZE THE CLONED UCB
(13)	1495	UNIT_INIT - INITIALIZE THE DEQNA UNIT
(14)	1598	FFI_INIT - FFI INTERFACE INITIALIZATION ROUTINE
(15)	1637	XMT_FDT - TRANSMIT I/O OPERATION FDT ROUTINE
(16)	1753	XMT_FFI_START - START FAST INTERFACE TRANSMIT OPERATION
(17)	1821	XMT_START - START TRANSMIT OPERATION
(19)	2196	RCV_FDT - RECEIVE I/O OPERATION FDT ROUTINE
(20)	2276	RCV_START - START RECEIVE I/O OPERATION
(21)	2357	SUBROUTINES TO FIND SHR DATA STRUCTURE
(22)	2419	ALT_START - ALTERNATE START I/O ROUTINE
(23)	2490	SETMODE_FDT - SET MODE I/O OPERATION FDT DISPATCH ROUTINE
(25)	2920	SHR_UCB - CREATE SHARED UCB
(26)	3061	CHECK_PARAM - CHECK SHARED USERS PARAMETERS
(27)	3123	SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
(28)	3155	SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
(29)	3276	READ_LINE_CTR - READ THE LINE COUNTERS
(29)	3277	READ_CIRC_CTR - READ THE CIRCUIT COUNTERS
(30)	3387	GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
(31)	3443	CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
(32)	3493	CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
(33)	3532	ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
(34)	3599	STARTIO - START I/O OPERATION
(35)	3689	START - START UNIT'S PROTOCOL
(36)	4215	SETUP_MODE - SETUP THE TRANSMIT BUFFER TO INIT QNA
(37)	4329	FILLRCVLIST - FILL RECEIVE BUFFER LIST
(38)	4408	START_RECEIVE - START ANY RECEIVE REQUESTS PENDING
(40)	4576.35	LOAD_RECV - LOAD CSR'S WITH RECEIVE RING ADDRESS IF NEEDED
(41)	4647	QNA_INTR - QNA INTERRUPT SERVICE ROUTINE
(42)	4709	SCHED_FORK - SCHEDULE THE FORK PROCESS
(42)	4710	SCHED_FORKC - SCHEDULE THE FORK PROCESS WITH R3 CLEAR
(43)	4754	FORK_PROC - Error and completion fork process handling
(44)	5054	RCV_ERROR - Process receive errors
(44)	5055	XMT_ERROR - Process transmit errors
(45)	5122	SUBROUTINES TO FIND SHR MATCH ON SOURCE ADDRESS
(46)	5184	COPY_RCV - Copy a receive buffer for the PROMISCUOUS user
(47)	5256	FINISH_XMT_FFI - Finish FAST interface transmit processing
(48)	5288	FINISH_RCV_FFI - Finish FAST receive processing
(49)	5334	FINISH_RCV_IO - Finish receive I/O processing
(50)	5473	ASSEM_PKTS - Assemble receive packets
(51)	5648	MOP_CTR_REQUEST - PROCESS MOP READ COUNTERS REQUEST
(52)	5693	MOP_CTR_BUILD - BUILD THE MOP COUNTER RETURN MESSAGE
(53)	5761	REG_DUMP - DEQNA ERROR LOG AND DIAGNOSTICS REGISTER DUMP
(54)	5795	RESTART_ROUT - PROCESS EXPIRATION OF RESTART TIMER
(55)	5885	TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
(56)	5962	TIMEOUT - TIMEOUT SERVICE ROUTINE
(57)	6006	ALLOC_CDB - ALLOCATE THE CDB
(58)	6072	SHUTDOWN_QNA - SHUTDOWN QNA AND ALL UNITS
(59)	6227	SHUTDOWN - SHUT DOWN UNIT
(59)	6228	SHUTDOWN_PROTYP - SHUT DOWN PROTOCOL TYPE
(60)	6497	BLD_STOP_IRP - Build an IRP to reset promiscuous mode
(61)	6544	BLD_STRT_IRP - Build a point-to-point startup IRP
(62)	6588	BLD_IRP - Build an IRP routine
(63)	6637	CLEANUP_SHR - CLEANUP ALL I/O ON SHARE DATA STRUCTURE
(63)	6638	DELETE_SHR - DELETE SHR DATA STRUCTURE
(64)	6766	CANCEL - CANCEL I/O ON UNIT
(65)	6965	SUBROUTINES TO FIND SHR DATA STRUCTURE GIVEN PCB AND CHAN
(66)	7022	FIND_POINT_UCB - Find the point to point UCB

(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

XC
VC


```

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 definitions
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 ;
0000 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 ;
0000 .1 POINT = 1 ; Enable use of point-to-point mode
0000 .2 ;
0000 .3 ; Define the following symbol to change the RING INVALIDATION test to
0000 .4 ; perform POLLing of the CSR's.
0000 .5 ;
0000 .6 POLL = 1 ; Enable POLL scanning of CSR's
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 ;

```

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG00C1
:RNG0001
-1

```

:RNG0001
-1

:RNG0001
:RNG0001
-4

```

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_PRT_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_CNFSIZ 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_CHAIN 1 : Maximum number of extra segments in
: a receive buffer chain
0000 235
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
0000 .1 $EQU INIT_C_QUOTA 6*MAX_PKT_SIZE : Allow for 6 of the largest buffers
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 DSCSA_POINTER 4 : Pointer to data in buffer descriptor
0000 245 $EQU MIN_PRT_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*1000*1000 : Delta interval (in 100 nsec)
0000 248 $EQU RESTART_DELTA 3*10000*1000 : RESTART interval - 3 Seconds
0000 .1 $EQU XMT_C_TIM 6 : XMTS must take less than 6 seconds
0000 .2 $EQU XMT_TIM <<XMT_C_TIM+2>/2>
0000 253 $EQU XQ_C_ADDRVC 64 : Size to add to received packets
0000 254 $EQU XQ_C_STPRO <^X0660> : X PT-TO-PT Starting protocol type 60-06
0000 255 $EQU IPLS_XQ_FIPL 8 : Fork IPL
0000 256 $EQU IPLS_XQ_DIPL 21 : Device IPL
0000 257
0000 258 :
0000 259 : Local macros
0000 260 :
0000 261 .MACRO SETBIT VAL,FLAG
0000 262 .NTYPE SS VAL
0000 263 .IF EQ Z SS_ -^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 SS VAL
0000 282 .IF EQ Z,SS,-^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 :
0000 337 :
0000 338 :
0000 339 :
0000 340 :
0000 341 :
0000 342 :
0000 343 :
0000 344 :
0000 345 :
0000 346 :
0000 347 :
0000 348 :
0000 349 :
0000 350 :
0000 351 :
0000 352 :
0000 353 :
0000 354 :
0000 355 :
0000 356 :
0000 357 :
0000 358 :
0000 359 :
0000 360 :
0000 361 :
0000 362 :
0000 363 :
0000 364 :
0000 365 :
0000 366 :
0000 367 :
0000 368 :
0000 369 :
0000 370 :
0000 371 :
0000 372 :
0000 373 :
0000 374 :
0000 375 :
0000 376 :
0000 377 :
0000 378 :
0000 379 :
0000 380 :
0000 381 :
0000 382 :
0000 383 :
0000 384 :
0000 385 :
0000 386 :
0000 387 :
0000 388 :
0000 389 :
0000 390 :
0000 391 :

```

```

TYPE = Parameter type
OFFSET = Offset in UCB/CDB to current value
WIDTH = Width of field in UCB/CDB (B,W,L)
MIN = Minimum value parameter is allowed to take
MAX = Maximum value parameter is allowed to take
INVALID = Invalid flags in status word
BASE = Data base (CDB,UCB)
STRING = Parameter is a string value
SIZE = Maximum size of string parameter in bytes
CHECK = Comparison is needed (YES,NO)

. IF BLANK type
. WORD 0
. IF FALSE ; BLANK type
$$$typ = type & prm_typ_m_code ; Isolate type code

$$$flg = 0
. IIF NOT_BLANK <invalid>, $$$flg = $$$flg!prm_flg_m_invalid
. IIF IDN <check><YES>, $$$flg = $$$flg!prm_flg_m_check
. IIF IDN <base><CDB>, $$$flg = $$$flg!prm_flg_m_cdb

. IF BLANK string
. WORD $$$typ
. IIF NOT_BLANK <min>, $$$flg = $$$flg!prm_flg_m_min
. IIF NOT_BLANK <max>, $$$flg = $$$flg!prm_flg_m_max
. BYTE $$$flg
$$$off = offset & prm_off_m_value ; Isolate offset only
$$$wid = 0 ; Set null width
. IIF IDN <width><B>, $$$wid = <1@prm_off_v_width>
. IIF IDN <width><W>, $$$wid = <2@prm_off_v_width>
. IIF IDN <width><L>, $$$wid = <3@prm_off_v_width>
. WORD $$$off!$$$wid
. IIF NOT_BLANK <min>, .WORD min
. IIF NOT_BLANK <max>, .WORD max
line_prm_bufsiz = line_prm_bufsiz + 6

. IF_FALSE ; BLANK STRING
. WORD $$$typ!prm_typ_m_string ; Indicate a string parameter
. BYTE $$$flg
$$$off = offset & prm_off_m_value ; Isolate offset only
$$$wid = <size @ prm_off_v_width> & prm_off_m_width ; Get max allowed
$$$siz = <$$$wid @ -prm_off_v_width>
. WORD $$$off!$$$wid
line_prm_bufsiz = line_prm_bufsiz + 4 + $$$siz

. ENDC ; BLANK STRING
. IIF NOT_BLANK <invalid>, .WORD invalid

. ENDC ; BLANK TYPE
. ENDM PARAM
=====

```

```

0000 392 .MACRO OFFSET 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 OFFSET
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>, OFFSET B,'offset','base
0000 419 .IIF IDN <width><16>, OFFSET W,'offset','base
0000 420 .IIF IDN <width><32>, OFFSET 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>, $$$map = 1@7
0000 433 .IF NOT BLANK <offset>
0000 434 .IIF IDN <width><8>, OFFSET B,'offset',LINE
0000 435 .IIF IDN <width><16>, OFFSET W,'offset',LINE
0000 436 .IIF IDN <width><32>, OFFSET L,'offset',LINE
0000 437 .BYTE $$$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 L:
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 + S
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 . = IRPSW FUNC+1  
0021 513 $DEF IRPSB_XQ_FUNC .BLKB 1 ; QNA driver internal function code  
0022 514  
00000038 0022 515 . = IRPSL MEDIA  
0038 516 $DEF IRPSW_XQ_RID ; RCV/XMT request ID  
0038 517 $DEF IRPSB_XQ_SLOT .BLKB 1 ; RCV/XMT mapping slot number  
0039 518 $DEF IRPSB_XQ_RING .BLKB 1 ; RCV/XMT ring entry number  
003A 519  
00000046 003A 520 . = IRPSQ STATION+6  
0046 521 $DEF IRPSB_XQ_DATAP .BLKB 1 ; XMT buffered data path number  
0047 522  
0000003C 0047 523 . = IRPSL MEDIA+4  
003C 524 $DEF IRPSL_XQ_SYSBUF .BLKL 1 ; XMT system buffer address  
0040 525  
0000003C 0040 526 . = IRPSL MEDIA+4  
003C 527 $DEF IRPSL_XQ_DATBUF .BLKL 1 ; User RCV data buffer address  
0040 528  
00000038 0040 529 . = IRPSL MEDIA  
0038 530 $DEF IRPSW_XQ_USERSIZ .BLKW 1 ; User P2 buffer size on sensemode  
003A 531  
0000003A 003A 532 . = IRPSL MEDIA+2  
003A 533 $DEF IRPSW_XQ_STATUS .BLKW 1 ; Completion status  
003C 534  
0000003C 003C 535 . = IRPSL MEDIA+4  
003C 536 $DEF IRPSL_XQ_USERBUF .BLKL 1 ; User P1 buffer address on sensemode  
0040 537  
00000040 0040 538 . = IRPSQ STATION  
0040 539 $DEF IRPSL_XQ_P2BUF .BLKL 1 ; User P2 buffer address on sensemode  
0044 540  
00000044 0044 541 . = IRPSQ STATION+4  
0044 542 $DEF IRPSW_XQ_P2SIZ .BLKW 1 ; P2 return buffer size on sensemode  
0046 543  
00000040 0046 544 . = IRPSQ STATION  
0040 545 $DEF IRPSW_XQ_CODE .BLKW 1 ; Bad parameter code on startup request  
0042 546  
0000003C 0042 547 . = IRPSL MEDIA+4  
003C 548 $DEF IRPSL_XQ_MAP .BLKL 1 ; Diagnostics buffer mapping info  
0040 549  
00000040 0040 550 . = IRPSQ STATION  
0040 551 $DEF IRPSL_XQ_DGUNI .BLKL 1 ; Diagnostics buffer UNIBUS address  
0044 552 $DEF IRPSL_XQ_UPADR .BLKW 1 ; Micro-process internal address  
0046 553  
00000094 0046 554 . = IRPSL RBUFH AD  
0094 555 $DEF IRPSL_XQ_SETUP .BLKL 1 ; Setup transmit buffer  
0098 556  
0000003A 0098 557 . = IRPSL MEDIA+2  
003A 558 $DEF IRPSW_XQ_PROTYP .BLKW 1 ; Protocol type for user  
003C 559  
00000090 003C 560 . = IRPSL LBOFF  
0090 561 $DEF IRPSL_XQ_SHR .BLKL 1 ; Address of SHR structure for user  
0094 562  
00000060 0094 563 . = IRPSL_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 _YIELD 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
0000020 0000 587  = CXBSL_SPARE1
0000022 0020 588  $DEF CXBSB_XQ_FUNC .BLKB 1 ; QNA driver internal function code
0000022 0021 589  .BLKB 1 ; SPARE
0022 590  $DEF CXBSW_XQ_RID ; RCV/XMT request ID
0022 591  $DEF CXBSB_XQ_SLOT .BLKB 1 ; RCV/XMT mapping slot number
0023 592  $DEF CXBSB_XQ_RING .BLKB 1 ; RCV/XMT ring entry number
0024 593
0024 594  ;
0024 595  ; The following overlays are for transmits only
0024 596  ;
0000024 0024 597  = CXBSL_SPARE0
0024 598
0024 599  ; NOTE: The following two fields area overlaped. So if the Low
0024 600  ; Bit is set, then the address is that of a UCB, else it's an IRP.
0024 601
0024 602  $DEF CXBSL_T_IRP ; Associated IRP address
0024 603  $DEF CXBSL_T_UCB .BLKL 1 ; Associated UCB address
0028 604
000003A 0028 605  = CXBSC_HEADER-<XQ_C_HEADER>
003A 606  $DEF CXBST_T_DATA .BLKB XQ_C_HEADER ; Standard Ethernet header
0048 607
0048 608  ;
0048 609  ; The following overlays are for receives only
0048 610  ;
000000B 0048 611  = CXBSB_CODE
000B 612  $DEF CXBSB_R_FLAGS .BLKB 1 ; Receive message flags
000C 613
000001C 000C 614  = CXBSL_END_ACTION
001C 615  $DEF CXBSW_R_NCHAIN .BLKW 1 ; Number of buffers in chain
001E 616
0000014 001E 617  = CXBSL_IRP
0014 618  $DEF CXBSW_R_STS .BLKW 1 ; Receive status
0016 619
0000038 0016 620  = CXBSC_HEADER - <XQ_C_HEADER+XQ_C_CNTSIZ>
0038 621  $DEF CXBST_R_DATA ; Start of receive data
0038 622  $DEF CXBSG_R_DEST .BLKW 3 ; Destination node address
003E 623  $DEF CXBSG_R_SRC .BLKW 3 ; Source node address
0044 624  $DEF CXBSW_R_PTYPE .BLKW 1 ; Protocol Type
0046 625  $DEF CXBST_R_USERDAT ; Start of user data
0046 626  $DEF CXBSW_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 (IRPSB_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
0000098 0000 640 . = UCBSQ_NI_LENGTH ; Position at end of UCB NI extension
0098 641
0098 642 $DEF UCBSQ_XQ_QUEUES ; Message and I/O request queue heads
0098 643 $DEF UCBSQ_XQ_SHARE .BLKQ 1 ; List of shared users
00A0 644 $DEF UCBSQ_XQ_IOQS ; Start of the I/O queues
00A0 645 $DEF UCBSQ_XQ_RCVMSG .BLKQ 1 ; Receive messages completed
00A8 646 $DEF UCBSQ_XQ_RCVREQ .BLKQ 1 ; Receive IRP waiting for messages
00B0 647 $DEF UCBSQ_XQ_XMTREQ .BLKQ 1 ; Xmit IRP wait queue (PT-TO-PT)
00000004 00B8 648 UCBSQ_XQ_QUEUES = <.-UCBSQ_XQ_QUEUES>/8 ; Number of queue heads
00B8 649
00B8 650 $DEF UCBSL_XQ_PID .BLKL 1 ; Starter's PID
00BC 651 $DEF UCBSL_XQ_CPID .BLKL 1 ; Creator's PID
00C0 652 $DEF UCBSL_XQ_AST .BLKL 1 ; Attention AST list
00C4 653 $DEF UCBSL_XQ_DEFUSR .BLKL 1 ; Default shared user (shared use only)
:RNG0001 00C8 1 $DEF UCBSL_XQ_CDB .BLKL 1 ; Address of CDB
00CC 654 $DEF UCBSW_XQ_QUOTA .BLKW 1 ; Receive buffer quota
00CE 655 $DEF UCBSW_XQ_PROTYP .BLKW 1 ; Ethernet protocol type
00D0 656
00D0 657 $DEF UCBSB_XQ_SETPRM ; Start of parameter section
00D0 658 $DEF UCBSG_XQ_DES .BLKW 3 ; Destination address for shared user
00D6 659 $DEF UCBSW_XQ_HBQ .BLKW 1 ; Hardware buffer quota
00D8 660 $DEF UCBSB_XQ_ACC .BLKB 1 ; Protocol access mode
00D9 661 $DEF UCBSB_XQ_BFN .BLKB 1 ; Number of receive buffers
00DA 662
00DA 663 $DEF UCBSB_XQ_SHRPRM ; Start of "shared user" validated prms
00DA 664 $DEF UCBSW_XQ_BSZ .BLKW 1 ; Device buffer size
00DC 665 $DEF UCBSB_XQ_PRO .BLKB 1 ; Protocol selection
00DD 666 $DEF UCBSB_XQ_PAD .BLKB 1 ; Padding mode
00DE 667 $DEF UCBSB_XQ_PRM .BLKB 1 ; Promiscuous mode
00DF 668 $DEF UCBSB_XQ_MLT .BLKB 1 ; Multicast (all) address state
00E0 669 $DEF UCBSB_XQ_DCH .BLKB 1 ; Data chaining on receives
00E1 670
00E1 671 $DEF UCBSB_XQ_CDBPRM ; Start of settable parameters for CDB
00E1 672 $DEF UCBSB_XQ_CON .BLKB 1 ; Controller mode
00000001 00E2 673 UCBSQ_XQ_CDBPRM = .-UCBSB_XQ_CDBPRM
00000008 00E2 674 UCBSQ_XQ_SHRPRM = .-UCBSB_XQ_SHRPRM
00E2 675
00000007 00E2 676 $DEF UCBSG_XQ_PHA .BLKW 3 ; User defined physical address
00E8 677 UCBSQ_XQ_SETPRM = .-UCBSB_XQ_CDBPRM
00E8 678
00E8 679 $DEF UCBSB_XQ_MST .BLKB 1 ; Maintenance state
00E9 680 $DEF UCBSB_XQ_MULT I .BLKB 1 ; Number of entries in MULTI
00EA 681 $DEF UCBSB_XQ_MLTTBL .BLKB 1 ; Number of entries in MLTTBL
00EB 1 $DEF UCBSB_XQ_SPARE .BLKB 1 ; SPARE BYTE
00EC 682 $DEF UCBSG_XQ_MULT I .BLKW 3*MAX_C_MLT ; Multicast address list
0134 683 $DEF UCBSG_XQ_MLTTBL .BLKW 3*MAX_C_MLT ; Multicast generation list
017C 684
017C 685 $DEF UCBSW_XQ_CTR ; Start of counter section
017C 686 $DEF UCBSW_XQ_MNECTR .BLKW 1 ; Multicast address not enabled
017E 687 $DEF UCBSW_XQ_UBUCTR .BLKW 1 ; No buffer available counter
0180 688 $DEF UCBSL_XQ_SBLCTR .BLKL 1 ; Number of blocks sent

```

```
0184 689 $DEF UCBSL_XQ_SBYCTR .BLKL 1 ; Number of bytes sent
0188 690 $DEF UCBSL_XQ_RBLCTR .BLKL 1 ; Number of blocks received
018C 691 $DEF UCBSL_XQ_RBYCTR .BLKL 1 ; Number of bytes received
0190 692 ; Unused/unneeded fields
0190 693 $DEF UCBSW_XQ_TOTQUO .BLKW 1 ; Total quota for shared UCB
0192 694 $DEF UCBSL_XQ_FFI .BLKL 1 ; Fast interface BLOCK address
0196 695 $DEF UCBSL_XQ_STIRP .BLKL 1 ; % Address of PT-TO-PT Startup IRP
019A 696 ;
019A 697 $DEF UCBSL_XQ_LENGTH ; Size of XQDRIVER UCB
019A 698 ;
019A 699 ;
019A 700 ; Define device status bits
019A 701 ;
019A 702 $VIELD UCB,0,<- ; XQDRIVER UCBSW_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 <,75,- ; Reserved
019A 711 <XQ_INTERLOCK,,M>,- ; RESTART bit is interlocked
019A 712 <XQ_RESTART,,M>,- ; Automatic RESTART on PROTOLOL
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 address (R/W)  
000E 733 $DEF CSR .BLKW 1 ; Port CSR  
0010 734  
0010 735 _YIELD 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 _YIELD 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 seqment
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 seqment
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

```



```
0000 875  
0000 876 :  
0000 877 : Block header for non-DECnet xmit buffers  
0000 878 :  
0000 879 $DEFINI BLK ; Define a standard block header  
0000 880  
0000 881 $DEF BLK_L_LINK .BLKL 2 ; Forward and backward queue links  
0008 882 $DEF BLK_W_SIZE .BLKW 1 ; Block size  
000A 883 $DEF BLK_B_TYPE .BLKB 1 ; Block type  
000B 884 $DEF BLK_B_SPARE .BLKB 1 ; SPARE byte  
000C 885 $DEF BLK_T_DATA ; Start of data  
000C 886 $DEF BLK_C_HEADER ; Size of buffer header  
000C 887  
000C 888 $DEFEND BLK
```

:RNG0001
-1

00000001

-1

00000005

:RNG0001
:RNG0001
:RNG0001
:RNG0001

```

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
0010 .1 $DEF CDB_L_LASTCSR ; Port CSR contents
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
001B 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
00E4 927 CDB_C_ABORTS = <.-CDB_Q_QUEUES>/8 ; Number of Queues to abort requests
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
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_RVCNT .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_UCBO .BLKL 1 ; Address of UCB #0
0114 .1 $DEF CDB_L_CSR .BLKL 1 ; DEQNA CSR address
0118 .2 $DEF CDB_B_SPARE .BLKB 1 ; SPARE BYTE
0119 .3 $DEF CDB_B_DIAG1 .BLKB 1 ; Diagnostic info byte
011A .4 $DEF CDB_W_DIAG2 .BLKW 1 ; Second word of diagnostic info

```

:RNG0001
:RNG0001
-3

```

011C      .5
011C      .6 $DEF      CDB_L_RINGMAP      .BLKL      1      : Mapping information for RINGS
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 failure
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
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
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)
020C      .1 $DEF      CDB_L_UV1BUF      .BLKL      1      : MicroVAX I buffer area address
0210      988 $DEF      CDB_L_PRMUSER      .BLKL      1      : Promiscuous user's UCB address
0214      989 $DEF      CDB_L_TQE      .BLKB      TQESC_LENGTH      : Timer queue element
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_MULT I      .BLKB      1      : Number of entries in Multicast list

```

-1

:RNG0001
-1


```

0000 1027 :
0000 1028 : P2 buffer header definition
0000 1029 :
0000 1030 $DEFINI P2B
0000 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
0000 1041 :
0000 1042 : Diagnostics buffer definition
0000 1043 :
0000 1044 $DEFINI DIAG
0000 1045 :
0000 1046 : Driver independent portion of diagnostics buffer
0000 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_G_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
0000 1077 :
0000 1078 : Receive buffer header definition
0000 1079 :
0000 1080 $DEFINI RHDR
0000 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 initied
002A 1117 >
002A 1118
002A 1119 $DEFEND SHR

```

```

0000 1121      .SBTTL Standard tables
0000 1122      :
0000 1123      : Driver prologue table
0000 1124      :
0000 1125      DPTAB -
0000 1126      END=XQ END,- ; END OF DRIVER
0000 1127      ADAPTER=UBA,- ; ADAPTER TYPE
0000 1128      UCBSIZE=UCBS$C_XQ_LENGTH,- ; SIZE OF UCB
0000 1129      NAME=XQDRIVER ; DRIVER NAME
0038 1130
0038 1131      DPT_STORE INIT ; START OF CONTROLLER INIT
0038 1132      DPT_STORE UCB,UCBS$B_FIPL,B,IPL$XQ_FIPL ; FORK IPL
003C 1133      DPT_STORE UCB,UCBS$B_DIPL,B,IPL$XQ_DIPL ; DEVICE IPL
0040 1134      DPT_STORE ORB,ORBS$B_FLAGS,B,- ; Protection block flags
0040 1135      ; ZORBSM PROT_16> ; SOGW protection word
0044 1136      DPT_STORE ORB,ORBS$W_PROT,Q,0 ; default protection
0049 1137      DPT_STORE ORB,ORBS$L_OWNER,L,0 ; no owner as yet
0050 1138      DPT_STORE UCB,UCBS$L_DEVCHAR,L,- ; DEVICE CHARACTERISTICS
0050 1139      <DEVS$M_SHR!-
0050 1140      DEVS$M_NET!-
0050 1141      DEVS$M_AVL!-
0050 1142      DEVS$M_IDV!-
0050 1143      DEVS$M_ODV>
0057 1144
0057 1145      DPT_STORE UCB,UCBS$B_DEVCLASS,B,DC$SCOM ; Device class
005B 1146      DPT_STORE UCB,UCBS$B_DEVTYPE,B,DT$DEQNA ; Device type
005F 1147      DPT_STORE UCB,UCBS$W_DEVBUFSIZ,W,ST2 ; Default buffer size
0064 1148      DPT_STORE UCB,UCBS$W_STS,W,<UCBS$M_ONLINE!UCBS$M_TEMPLATE>
0069 1149      DPT_STORE UCB,UCBS$G_XQ_PHA,L,-1 ; No default physical address
0070 1150      DPT_STORE UCB,UCBS$G_XQ_PHA+4,W,-1 ; ...
0075 1151      :
0075 1152      : Store defaults for all parameters
0075 1153      :
0075 1154      DPT_STORE UCB,UCBS$W_XQ_BSZ,W,MAX_PKT_SIZE ; Default device buffer size
007A 1155      DPT_STORE UCB,UCBS$B_XQ_BFN,B,1 ; Default user buffer number
007E 1156      DPT_STORE UCB,UCBS$W_XQ_HBQ,W,INIT C QUOTA ; Hardware Buffer Quota
0083 1157      DPT_STORE UCB,UCBS$B_XQ_PRO,B,NMASC_CINPR_NI ; 'NI' is the protocol mode
0087 1158      DPT_STORE UCB,UCBS$B_XQ_PRM,B,NMASC_STATE_OFF ; Promiscuous mode is OFF
008B 1159      DPT_STORE UCB,UCBS$B_XQ_MLT,B,NMASC_STATE_OFF ; All multicasts is OFF
008F 1160      DPT_STORE UCB,UCBS$B_XQ_DCH,B,NMASC_STATE_ON ; Data chaining is ON
0093 1161      DPT_STORE UCB,UCBS$B_XQ_PAD,B,NMASC_STATE_ON ; Padding is ON
0097 1162      DPT_STORE UCB,UCBS$B_XQ_CON,B,NMASC_LINCN_NOR ; Controller mode is NORMAL
009B 1163      DPT_STORE UCB,UCBS$B_XQ_ACC,B,NMASC_ACC_EXC ; Exclusive mode is default
009F 1164
009F 1165      DPT_STORE REINIT
009F 1166
009F 1167      DPT_STORE DDB,DBS$L_DDT,D,XQ$DDT ; DDT ADDRESS
00A4 1168      DPT_STORE CRB,CRBS$L_INTD+4,D,QNA_INTR ; QNA interrupt service routine
00A9 1169      DPT_STORE CRB,CRBS$L_INTD+VEC$L_INITIAL,D,CONTROL_INIT ; CONTROLLER INIT ADDRE
00AE 1170      DPT_STORE CRB,CRBS$L_INTD+VEC$L_UNITINIT,D,UNIT_INIT ; UNIT INIT
00B3 1171      DPT_STORE CRB,CRBS$L_INTD+VEC$L_START,D,FFI_INIT ; FFI INIT
00B8 1172      DPT_STORE END
0000 1173
0000 1174      .PSECT $$$115_DRIVER, LONG
0000 1175      :
0000 1176      : LOCAL STORAGE
0000 1177      :

```

:RNG0001
-1

```

0000 1178 : Driver dispatch table
0000 1179 :
0000 1180 : DDTAB DEVNAM=XQ,- : DRIVER DISPATCH TABLE
0000 1181 : START=STARTIO,- : Start I/O operation
0000 1182 : FUNCTB=XQ FUNCTABLE,- : Function decision table address
0000 1183 : CANCEL=CANCEL,- : CANCEL I/O entry point
0000 1184 : REGDMP=REG_DUMP,- : Register dump entry point
0000 1185 : DIAGBF=<DIAG_C LENGTH>,- : Diagnostic buffer size
0000 1186 : CLONEDUCB=CLONED_UCB,- : Cloned UCB initialization
0000 1187 : ALTSTART=ALT_START : Alternate start I/O entry point
0038 1188 :
0038 1189 : Function decision table
0038 1190 :
0038 1191 : XQ_FUNCTABLE:
0038 1192 : FUNCTAB ,- : Legal Functions
0038 1193 : <WRITEVBLK,WRITELBLK,WRITEPBLK,READVBLK,READLBLK,-
0038 1194 : READPBLK,SENSEMODE,SENSECHAR,SETMODE,SETCHAR>
0040 1195 : FUNCTAB ,- : Buffered Functions
0040 1196 : <WRITEVBLK,WRITELBLK,WRITEPBLK,READVBLK,READLBLK,-
0040 1197 : READPBLK,SENSEMODE,SENSECHAR,SETMODE,SETCHAR>
0048 1198 : FUNCTAB XMT_FDT,<WRITELBLK,WRITEPBLK,WRITEVBLK> ;
0054 1199 : FUNCTAB RCV_FDT,<READLBLK,READPBLK,READVBLK> ;
0060 1200 : FUNCTAB SETMODE_FDT,<SETMODE,SETCHAR> ;
006C 1201 : FUNCTAB SENSEMODE_FDT,<SENSEMODE,SENSECHAR> ;
0078 1202 :

```

```

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  NMASC_PCLI_HBQ,-    ; Hardware Buffer Quota
0078 1244          OFFSET=UCBSW_XQ_HBQ,-
0078 1245          WIDTH=W,MAX=T6384,-
0078 1246          INVALID=UCBSM_XQ_INITED
0081 1247
0081 1248 LINE_PARAM:                    ; Start of line parameters
0081 1249
0081 1250      PARAM  NMASC_PCLI_ACC,-      ; Access mode for protocol type
0081 1251          OFFSET=UCBSB_XQ_ACC,-
0081 1252          WIDTH=B,-
0081 1253          MIN=NMASC_ACC_SHR,-
0081 1254          MAX=NMASC_ACC_EXC
008A 1255
008A 1256      PARAM  NMASC_PCLI_PRO,-      ; Protocol selection mode
008A 1257          OFFSET=UCBSB_XQ_PRO,-
008A 1258          WIDTH=B,-
008A 1259          MIN=NMASC_LINPR_POI,- ; % Accept either point or NI
008A 1260          MAX=NMASC_LINPR_NI

```

0093	1261		
0093	1262	PARAM	NMASC_PCLI_BUS,- ; Buffer size
0093	1263		OFFSET=UCBSW_DEVBUSIZ,-
0093	1264		WIDTH=W,-
0093	1265		MIN=MIN_PKT_SIZE,- ; User buffer LIMITS
0093	1266		MAX=MAX_PKT_SIZE,-
0093	1267		INVALID=UCBSM_XQ_INITED
009E	1268		
009E	1269	PARAM	NMASC_PCLI_BFN,- ; Buffer number
009E	1270		OFFSET=UCBSB_XQ_BFN,-
009E	1271		WIDTH=B,-
009E	1272		MIN=0,MAX=255,-
009E	1273		INVALID=UCBSM_XQ_INITED
00A9	1274		
00A9	1275	PARAM	NMASC_PCLI_PHA,- ; Physical NI address
00A9	1276		OFFSET=UCBSG_XQ_PHA,-
00A9	1277		STRING=YES,-
00A9	1278		SIZE=<2+6>,-
00A9	1279		INVALID=UCBSM_XQ_INITED
00B0	1280		
00B0	1281	PARAM	NMASC_PCLI_DCH,- ; Data chaining on receives
00B0	1282		OFFSET=UCBSB_XQ_DCH,-
00B0	1283		WIDTH=B,-
00B0	1284		MAX=NMA\$C_STATE_OFF
00B7	1285		
00B7	1286	PARAM	NMASC_PCLI_PAD,- ; Padding mode
00B7	1287		OFFSET=UCBSB_XQ_PAD,-
00B7	1288		WIDTH=B,-
00B7	1289		MAX=NMA\$C_STATE_OFF
00BE	1290		
00BE	1291	PARAM	NMASC_PCLI_PRM,- ; Promiscuous mode state
00BE	1292		OFFSET=UCBSB_XQ_PRM,-
00BE	1293		WIDTH=B,-
00BE	1294		MAX=NMA\$C_STATE_OFF,-
00BE	1295		INVALID=UCBSM_XQ_INITED
00C7	1296		
00C7	1297	PARAM	NMASC_PCLI_MLT,- ; Accept all multicast addresses
00C7	1298		OFFSET=UCBSB_XQ_MLT,-
00C7	1299		WIDTH=B,-
00C7	1300		MAX=NMA\$C_STATE_OFF,-
00C7	1301		INVALID=UCBSM_XQ_INITED
00D0	1302		
00D0	1303	PARAM	NMASC_PCLI_CON,- ; Controller mode
00D0	1304		OFFSET=UCBSB_XQ_CON,-
00D0	1305		WIDTH=B,-
00D0	1306		MAX=NMA\$C_LINCN_LOO,-
00D0	1307		INVALID=UCBSM_XQ_INITED
00D9	1308		
00D9	1309	PARAM	NMASC_PCLI_PTY,- ; Protocol type
00D9	1310		OFFSET=UCBSW_XQ_PROTYP,-
00D9	1311		WIDTH=W,-
00D9	1312		INVALID=UCBSM_XQ_INITED
00E0	1313		
00E0	1314	PARAM	NMASC_PCLI_MCA,- ; Multicast address list
00E0	1315		OFFSET=UCBSG_XQ_MULT1,-
00E0	1316		STRING=YES,-
00E0	1317		SIZE=2+<6*MAX_C_MLT> ; Maximum size of list

```

00E5 1318
00E5 1319      PARAM  NMASC PCLI BSZ,-      : Device buffer size
00E5 1320      OFFSET=UCBSW_XQ_BSZ,-
00E5 1321      WIDTH=W,-
00E5 1322      MIN=MIN_PKT_SIZE,-
00E5 1323      MAX=MAX_PKT_SIZE,-
00E5 1324      INVALID=UCBSM_XQ_INITED
00F0 1325
00F0 1326      PARAM  NMASC PCLI DES,-      : Destination Address for shared
00F0 1327      OFFSET=UCBSG_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  NMASC PCLI CRC,-      : CRC enabled
00F5 1335      OFFSET=UCBSB_XQ_MST,-      : garbage
00F5 1336      WIDTH=B,-
00F5 1337      MAX=NMASC_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  NMASC PCCI MST,-      : Maintenance state
00FE 1344      OFFSET=UCBSB_XQ_MST,-
00FE 1345      WIDTH=B,-
00FE 1346      MIN=NMASC_STATE_ON,-
00FE 1347      MAX=NMASC_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:
0109 1355      COUNTER  ZER, 16, ZERO      : Start of LINE counters
0109 1356      COUNTER  DBR, 32, DBRCTR   : Seconds since last zeroed
0111 1357      COUNTER  MBL, 32, MBLCTR   : Packets received
0115 1358      COUNTER  RFL, 16, RFLMAP,, MAP : Multicast packets received
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:
0159 1377 COUNTER DBS, 32, SBLCTR, CIRC ; Start of CIRCUIT counters
015D 1378 COUNTER BSN, 32, SBYCTR, CIRC ; Blocks sent
0161 1379 COUNTER DBR, 32, RBLCTR, CIRC ; Bytes sent
0165 1380 COUNTER BRC, 32, RBYCTR, CIRC ; Blocks received
0169 1381 COUNTER MNE, 16, MNECTR, CIRC ; Bytes received
016D 1382 COUNTER UBU, 16, UBUCTR, CIRC ; Multicast address not enabled
0171 1383
0171 1384 :
0171 1385 : MOP read counters return table (in order of COUNTERs returned)
0171 1386 :
0171 1387 MOPCTR TAB:
0171 1388 MOPCTR 16, ZERO ; Start of MOP counters
0174 1389 MOPCTR 32, BRCCTR ; Seconds since last zeroed
0177 1390 MOPCTR 32, BSNCTR ; Bytes received
017A 1391 MOPCTR 32, DBRCTR ; Bytes transmitted
017D 1392 MOPCTR 32, DBSCTR ; Packets received
0180 1393 MOPCTR 32, MBYCTR ; Packets transmitted
0183 1394 MOPCTR 32, MBLCTR ; Multicast bytes received
0186 1395 MOPCTR 32, BIDCTR ; Multicast packets received
0189 1396 MOPCTR 32, BS1CTR ; Packets transmitted - deferred
018C 1397 MOPCTR 32, BSMCTR ; Packets transmitted - 1 error
018F 1398 MOPCTR 16, SFLCTR, MAP ; Packets transmitted - several errors
0192 1399 MOPCTR 16, RFLCTR, MAP ; Transmit packets aborted
0195 1400 MOPCTR 16, UFDCTR ; Packets received in error
0198 1401 MOPCTR 16, OVRCTR ; Unrecognized frame destination
019B 1402 MOPCTR 16, LBECTR ; Receives lost - Internal buffer error
019E 1403 MOPCTR 16, UBUCTR ; Receives lost - Local buffer error
01A1 1404 MOPCTR 16, CDCCTR ; User buffer unavailable
01A4 1405 MOPCTR ; Transmit collision check failure
01A6 1406 : ; End of table
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 UCBSB_XQ_CON EQ UCBSB_XQ_CDBPRM
0456 01A6 1421 BAD_PARAM_TBL:
01A8 1422 .WORD NMASC_PCLI_CON
01A8 1423
01A8 1424 ASSUME UCBSW_XQ_BSZ EQ UCBSB_XQ_SHRPRM
01A8 1425 ASSUME UCBSB_XQ_PRO EQ UCBSW_XQ_BSZ+2
01A8 1426 ASSUME UCBSB_XQ_PAD EQ UCBSB_XQ_PRO+1
01A8 1427 ASSUME UCBSB_XQ_PRM EQ UCBSB_XQ_PAD+1
01A8 1428 ASSUME UCBSB_XQ_MLT EQ UCBSB_XQ_PRM+1
01A8 1429 ASSUME UCBSB_XQ_DCH EQ UCBSB_XQ_MLT+1
0B1B 01A8 1430 ASSUME UCBSB_XQ_CDBPRM EQ UCBSB_XQ_DCH+1
01A8 1431 .WORD NMASC_PCCI_DCH

```

XQDRIVER
V04-001

- VAX/VMS QNA driver
Local driver storage

K 9

8-JAN-1985 17:49:06 VAX/VMS Macro V04-00 Page 28
5-SEP-1984 00:20:54 [DRIVER.BUGSRC]XQDRIVER.MAR;1 (10)

0B19	01AA	1432	.WORD	NMASC_PCLI_MLT
0B18	01AC	1433	.WORD	NMASC_PCLI_PRM
0B1A	01AE	1434	.WORD	NMASC_PCLI_PAD
0458	01B0	1435	.WORD	NMASC_PCLI_PRO
0B20	01B2	1436	.WORD	NMASC_PCLI_BSZ
0B20	01B4	1437	.WORD	NMASC_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      :
05 01B6 1462 CONTROL_INIT::      : Initialize the DEQNA
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 = $$$_NORMAL
01B7 1477      :     R2 = UCB address of cloned UCB
01B7 1478      :     R3 = DDT address
01B7 1479      :     R4 = PCB address
01B7 1480      :     R5 = UCB address of template UCB
01B7 1481      :
01B7 1482      :     IPL = ASTDEL
01B7 1483      :
01B7 1484      : Outputs:
01B7 1485      :
01B7 1486      :     R0 = $$$_NORMAL
01B7 1487      :     R5 = UCB address of cloned UCB
01B7 1488      :     All other registers and IPL are preserved.
01B7 1489      :--
01B7 1490      :
55  52  D0 01B7 1491 CLONED_UCB::      : Cloned UCB initialization
01B7 1492      :     MOVL   R2,R5      : Copy UCB address
01BA 1493      :

```

```

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::
01BA 1519      : :&& JSB G^INISBRK ;*** TEMP *** ; Initialize a DEQNA unit
01BA 1520      :
01BA 1521      : PUSH R0,R1,R2,R3,R4,R5 ; Save all regs
19 64 A5 3F BB 01BA 1521      : PUSH R0,R1,R2,R3,R4,R5 ; Save all regs
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, UCBSL_NI_MLTPTR) to the multicast address table for
01C1 1527      : this protocol type.
01C1 1528      :
01C1 1529      :
00EC C5 9E 01C1 1530      : MOVAB UCBSG_XQ_MULTI(R5),- ; Initialize the pointer to this
0094 C5 01C5 1531      : UCBSL_NI_MLTPTR(R5) ; protocol's multicast address table
01C8 1532      :
01C8 1533      : MOVL #UCB$C_XQ_QUEUES,R0 ; Get number of queue listheads in UCB
51 50 04 D0 01C8 1534      : MOVAB UCBSQ_XQ_QUEUES(R5),R1 ; Get address of queue listheads
0098 C5 9E 01CB 1534      : MOVAB UCBSQ_XQ_QUEUES(R5),R1 ; Get address of queue listheads
81 81 61 DE 01D0 1535      : MOVAL (R1),R1+ ; Set forward link pointer
FC A1 D0 01D3 1536      : MOVL -4(R1),(R1)+ ; Set backward link pointer
F6 50 F5 01D7 1537      : SOBGTR R0,10$ ; Loop if more listheads
01DA 1538      :
01DA 1539      : 15$: TSTW UCBSW_UNIT(R5) ; Is this unit 0?
54 A5 B5 01DA 1539      : TSTW UCBSW_UNIT(R5) ; Is this unit 0?
15 13 01DD 1540      : BEQL 17$ ; Br if yes - Leave TEMPLATE bit on
0082 C5 B4 01DF 1541      : CLRW UCBSW_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 UCBSL_DDB(R5),R0 ; Get address of QNA DDB
50 04 A0 D0 01E7 1547      : MOVL DDB$UCB(R0),R0 ; Get address of UNIT 0 UCB
04 04 E0 01EB 1548      : BBS #UCB$ONLINE,- ; Br if QNA is ONLINE
04 64 A0 01ED 1549      : UCBSW_STS(R0),17$
64 10 AA 01F0 1550      : BICW #UCB$ONLINE,- ; Else, mark new unit as OFFLINE
64 A5 01F2 1551      : UCBSW_STS(R5)

```



```

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::
026A 1624      PUSHR  #M<R1,R2,R3,R4,R5>      ; Save registers
026C 1625      CLRL   R0                      ; Assume failure
55   34  A4  D0  026E 1626      MOVL   FFI$ DL UCB(R4),R5      ; Get UCB address
OE   68  A5  E1  0272 1627      BBC    #UCB$ XQ RUN,-        ; Br if device not ready
0346  CF  9E  0274 1628      MOVB  UCBSW DEVSTS(R5),90$
10   A4  0277 1629      MOVAB W^XMT_FFI_START,-    ; Return address of XMIT routine
0192  C5  54  D0  027B 1630      MOVL  FFI$ XMIT(R4)
50   01  9A  027D 1631      MOVL  R4,UCB$ XQ_FFI(R5)   ; Save FFI address
0192  C5  54  9A  0282 1632      MOVZBL #1,R0              ; Return success
50   01  BA  0285 1633 90$: POPR  #M<R1,R2,R3,R4,R5>   ; Restore registers
0192  C5  54  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 :--
00B5 31 0288 1672 ABORTIO_BR: ABORTIO ; Long branch to ABORTIO
0288 1673 -BRW ABORTIO ; Abort the I/O request
0288 1674
0288 1675 XMT_FDT:: ; Transmit FDT routine
40 A3 7C 0288 1676 CLRQ IRP$Q_STATION(R3) ; Zero the destination address
00CE C5 B0 028E 1677 MOVW UCBSW_XQ_PROTYP(R5),- ; Assume we are a non-promiscuous user
3A A3 0292 1678 IRP$W_XQ_PROTYP(R3)
51 10 AC D0 0294 1679 MOVL P5(APT,RT) ; 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 :
12 68 03 E0 029A 1686 BBS #UCBSV_XQ_SHARE,- ; Br if shared user
50 0C 9A 029C 1687 UCBSW_DEVSTS(R5),20$
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

```

```

02H1 1694 ASSUME NMAC_STATE_OFF EQ 1
02I1 1695
OD 00DE C5 E8 02B1 1696 20$: BLBS UCBSB_XQ_PRM(R5),30$ ; Br if user is not promiscuous
51 06 C0 02B5 1697 ADDL #6,R1 ; Point to protocol type
3A A3 61 B0 02B5 1698 IFNORD #2,(R1),ABORTIO BR ; Check access to buffer
50 14 9A 02BF 1699 MOVW (R1),IRPSW_XQ_PROTYP(R3) ; Get protocol type from user P5 buffer
58 6C D0 02C3 1700 30$: MOVZBL S^#SS$BADPARAM,R0 ; Assume bad parameters
59 04 AC 3C 02C6 1701 MOVL P1(AP),R8 ; Get starting address of user buffer
50 71 13 02C9 1702 MOVZWL P2(AP),R9 ; Get length of user buffer
00000000'GF 7D 02CD 1703 BEQL ABORTIO ; Br if zero length buffer
02D2 1704 MOVQ R8,R0 ; Retrieve buffer parameters
02D8 1705 JSB G^EXESWRITECHK ; Check accessibility of user buffer
02D8 1706 ; (No return on NO ACCESS)
2E 51 B1 02D8 1707 ; Returns IRPSW_BCNT
03 1E 02DB 1709 CMPW R1,#MIN_PKT_SIZE ; Is buffer at least minimum?
51 2E 3C 02DD 1710 BGEQU 50$ ; Br if yes, okay
00000048 8F C0 02E0 1711 50$: ADDL2 #CXBS_C_HEADER,R1 ; Else, allocate minimum sized packet
51 C2E6 ; Calculate length of buffer needed
00000000'GF 38 BB 02E7 1712 PUSHR #^M<R3,R4,R5> ; Save registers
4C 50 E9 02E9 1713 JSB G^EXESBUFRQUOTA ; Check if process has sufficient quota
00000000'GF 16 02EF 1714 BLBC R0,90$ ; Br if quota check failure
43 50 E9 02F2 1715 JSB G^EXESALLOCBUF ; Allocate CXB buffer for output
50 53 6E D0 02F8 1716 BLBC R0,90$ ; If LBC allocation failure
20 0080 C4 D0 02FB 1717 MOVL (SP),R3 ; Retrieve address of IRP
30 A3 51 C2 02FE 1718 MOVL PCB$JIB(R4),R0 ; Get JIB address
2C A3 52 B0 0303 1719 SUBL R1,JIB$BYTCNT(R0) ; Adjust buffered I/O quota
52 DD 0307 1720 MOVW R1,IRPSW_BOFF(R3) ; Set number of bytes charged to quota
030B 1721 MOVL R2,IRPSW_SVAPE(R3) ; Save CXB address in IRP
030F 1722 PUSHL R2 ; Save pointer to CXB
0311 1723
0311 1724 ASSUME CXBSL_FL EQ 0
0311 1725 ASSUME CXBSL_BL EQ CXBSL_FL+4
82 7C 0311 1726 CLRQ (R2)+ ; Clear link cells
0313 1727
82 51 B0 0313 1728 ASSUME CXBSW_SIZE EQ CXBSL_BL+4
0313 1729 MOVW R1,(R2)+ ; Set size of structure
0316 1730
0316 1731 ASSUME CXBSB_TYPE EQ CXBSW_SIZE+2
0316 1732 ASSUME CXBSB_CODE EQ CXBSB_TYPE+1
82 1B 9B 0316 1733 MOVZBW #DYN$C_CXB,(R2)+ ; Set structure type
52 6E D0 0319 1734 MOVL (SP),R2 ; Get back CXB address
031C 1735
031C 1736 ASSUME CXB$C_HEADER EQ CXB$T_DATA+XQ_C_HEADER
18 A2 3A B0 031C 1737 MOVW #CXB$T_DATA,CXBSW_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 MOV C3 R9,(R8),(R2) ; Move data to system buffer
3C BA 0328 1742 POPR #^M<R2,R3,R4,R5> ; Restore registers
53 DD 032A 1743 SETIPL UCBSB_FIPL(R5) ; Sync access to UCB
57 10 032E 1744 PUSHL R3 ; Save IRP address
53 8ED0 0330 1745 BSBB XMT_START ; Do common processing
08 50 E9 0332 1746 POPL R3 ; Restore IRP address
00000000'GF 17 0335 1747 BLBC R0,ABORTIO ; Br if error in processing request
0338 1748 JMP G^EXESQIORETURN ; Exit QIO service to await completion
033E 1749

```

XQDRIVER
V04-001

- VAX/VMS QNA driver
XMT_FDT - TRANSMIT I/O OPERATION FDT ROU

F 10

8-JAN-1985 17:49:06 VAX/VMS Macro V04-00 Page 36
5-SEP-1984 00:20:54 [DRIVER.BUGSRC]XQDRIVER.MAR;1 (15)

00000000'GF 38 BA 033E 1750 90\$: POPR #*M<R3,R4,R5> ; Restore registers
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 FFISL_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::
0346 1788      PUSH  #M<R1,R2,R4,R5>      ; Start FAST interface transmit request
55  34  A4  BB  0348 1789      MOVL  FFISL_DL_UCB(R4),R5      ; Save registers
                                ; Get UCB address
034C 1790      :
034C 1791      : For the FFI Interface, we will save the UCB address
034C 1792      :
24  A3  55  DO  034C 1793      MOVL  R5,CXB$T_UCB(R3)      ; Save UCB address
0350 1794      ASSUME CXB$T_UCB EQ CXB$T_IRP
24  A3  01  88  0350 1795      BISB  #1,CXB$C_T_UCB(R3)      ; ...indicate this is a UCB address
0354 1796      :
18  A3  0E  A2  0354 1797      SUBW  #XQ_C_HEADER,CXB$W_BOFF(R3) ; Back up offset for Ethernet header
0358 1798      :
0358 1799      ASSUME  NMAC_STATE_ON EQ 0
0358 1800      ASSUME  NMAC_STATE_OFF EQ 1
04  00DD C5  E8  0358 1801      BLBS  UCBSB_XQ_PAD(R5),20$      ; 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$W_BOFF(R3),R1      ; Get offset to start of data
52  53  51  C1  0365 1804      ADDL3 R1,R3,R2      ; Set R2 to start of header
0369 1805      :
0369 1806      ASSUME  XBUF_G_SRC EQ XBUF_G_DEST+6
28  A3  7D  0369 1807      MOVQ  CXB$Q_STATION(R3),=-      ; Store destination address
036C 1808      XBUF_G_DEST(R2)
036D 1809      ASSUME  XBUF_Q_TYPE EQ XBUF_G_SRC+6

```

```
00E2 C5 7D 036D 1810      MOVQ   UCBSG_XQ_PHA(R5),-      ; Store our source address
      06 A2              0371 1811      XBUF-G_SRC(R2)
00CE C5 B0 0373 1812      MOVW   UCBSW_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   UCBSL_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 :
0389 1854 : .ENABL LSB
0389 1855 : XMT_START:: : Start transmit operation
0389 1856 : BBS #UCB$V_XQ_RUN,- : Br if unit is in RUN mode
0389 1857 : UCB$W_DEVSTS(R5),30$
0389 1858 10$: MOVZWL #SS$ DEVINACT,R0 : Assume unit not started yet
0389 1859 : BBC #UCB$V_XQ_INTERLOCK,- : Br if unit is not re-starting
0389 1860 : UCB$W_DEVSTS(R5),20$ : on it's own.
0389 1861 : MOVZWL #SS$_OPINCOMPL,R0 : Else, return different error code
0389 1862 20$: RSB : Okay to leave now
0389 1863 :
0389 1864 30$: MOVL R3,CXB$T_IRP(R2) : Save IRP address
0389 1865 : CLRL IRP$L_XQ_SETUP(R3) : Indicate no SETUP buffer present
0389 1866 : MOVW IRP$W_BCNT(R3),CXB$W_BCNT(R2) : Set BCNT in CXB
0389 1867 : MOVQ R2,R3 : Copy IRP, CXB addresses
0389 1868 :
0389 1869 :
0389 1870 : ASSUME NMASC_STATE_ON EQ 0
0389 1871 : ASSUME NMASC_STATE_OFF EQ 1
0389 1872 : BLBS UCB$B_XQ_PAD(R5),40$ : Br if padding is disabled
0389 1873 : SUBW #XQ_C_CNTSIZ,CXB$W_BOFF(R3) : Else, skip length field of buffer
0389 1874 40$: MOVZWL CXB$W_BOFF(R3),R1 : Get offset to start of data
0389 1875 : ADDL3 R1,R3,R2 : Set R2 to start of header
0389 1876 : ASSUME XBUF_G_SRC EQ XBUF_G_DEST+6
0389 1877 : MOVQ IRP$Q_STATION(R4),- : Store destination address
0389 1878 : XBUF_G_DEST(R2)

```

```

04 E0
10 68 A5
50 20D4 8F 3C
05 68 A5 0E E1
50 02D4 8F 3C
05
24 A2 53 D0
0094 C3 D4
1A A2 32 A3 B0
53 52 7D
04 00DD C5 E8
18 A3 02 A2
51 18 A3 3C
52 53 51 C1
40 A4 7D
62 03C2

```

:RNG0001
-2

```

00E2 C5 7D 03C3 1879 ASSUME XBUF_W_TYPE EQ XBUF_G_SRC+6
06 A2 03C3 1880 MOVQ UCBSG_XQ_PHA(R5),- ; Store our source address
3A A4 B0 03C7 1881 XBUF_G_SRC(R2)
OC A2 03C9 1882 MOVW IRPSW_XQ_PROTYP(R4),- ; Store PROTOCOL TYPE
03CC 1883 XBUF_W_TYPE(R2)
03CE 1884
03CE 1885 XMT_INITIATE:: ; FAST Interface entry point (FFI)
03CE 1886
03CE 1887
03CE 1888 inputs:
03CE 1889 R5 = UCB address
03CE 1890 R3 = CXB address
03CE 1891 R2 = Start address for Ethernet header
20 A3 01 90 03CE 1892 MOVB #XQ_FC_V_XMIT,CXBSB_XQ_FUNC(R3) ; Set function request in CXB
03D2 1893
03D2 1894 ASSUME NMA$C_STATE_ON EQ 0
09 00DD C5 E8 03D2 1895 ASSUME NMA$C_STATE_OFF EQ 1
03D7 1896 BLBS UCBSB_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.
1A A3 B0 03D7 1901 MOVW CXBSW_BCNT(R3),- ; Else, store size of data-only
OE A2 03DA 1902 XBUF_Q_SIZE(R2) ; portion of buffer in message
1A A3 02 A0 03DC 1903 ADDW #XQ_C_CNTRSZ,CXBSW_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$: CMPW CXBSW_BCNT(R3),#MAX_PKT_SIZE ; Is buffer size bigger than
05DC 8F 03E3
03E6 1908 ; largest Ethernet buffer allowed?
50 034C 8F 1B 03E6 1909 BLEQU 90$ ; Br if no
3C 03E8 1910 80$: MOVZWL #SS$_IVBUFLN,R0 ; Assume bad buffer length
05 03ED 1911 RSB ; ELSE, leave now
03EE 1912
54 00C8 C5 D0 03EE 1913 90$: MOVL UCBSL_XQ_CDB(R5),R4 ; Get CDB address
01 E1 03F3 1915 BBC #CDB_STS-V_RUN,- ; Br if QNA not running
1B 0246 C4 03F5 1916 CDB_B_STS(R4),100$
OE A0 03F9 1917 ADDW #XQ_C_HEADER,- ; Adjust byte count
1A A3 03FB 1918 CXBSQ_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!
4E 68 A5 E1 03FD 1925 BBC #UCBSV_XQ_SHARE,- ; Br if NOT a shared user
03FF 1926 UCBSW_DEVSTS(R5),NO_SHR ;
0402 1927
0402 1928 ; Try to find a match on PID/CHAN. Returns pointer in R1
0402 1929
0402 1930
OE 24 A3 E8 0402 1931 ASSUME CXBSL_T_IRP EQ CXBSL_T_UCB
53 53 DD 0406 1932 BLBS CXBSL_T_IRP(R3),100$ ; Br if FFI user, return failure
24 A3 D0 0408 1933 PUSHL R3 ; Else, save CXB address
0301 30 040C 1934 MOVL CXBSL_T_IRP(R3),R3 ; Get IRP address
06 13 040F 1935 BSBW MATCH_SHR ; Try to find the SHR data structure
BEQL 110$ ; Br if match

```

:RNG0001
-2

```

      53 BED0 0411 1936          POPL   R3          ; Restore CXB address
      FF77 31 0414 1937 100$: BRW    10$          ; Else, error
      53 BED0 0417 1938
      00C4 C5 D1 041A 1940 110$: POPL   R3          ; Restore CXB address
      OE 13 041F 1941          CMPL   UCBSL_XQ_DEFUSR(R5),R1 ; Is this the default user?
      0421 1942          BEQL   SHR_DEF          ; Br if yes
      0421 1943          ; This is a SHARED-LIMITED user.
      0421 1944          ;
      2C 62 EB 0421 1945          BLBS   XBUF_G_DEST(R2),NO_SHR ; Br if multicast address
      12 A1 D0 0424 1946          MOVL  SHR_G_DEST(R1),-      ; Else, get destination from SHR struct.
      62 0427 1947          XBUF_G_DEST(R2)
      16 A1 B0 0428 1948          MOVW  SHR_G_DEST+4(R1),-    ;
      04 A2 11 042B 1949          XBUF_G_DEST+4(R2)
      21 042D 1950          BRB    NO_SHR          ;
      042F 1951          .DSABL LSB          ; Continue in common code
      042F 1952          ;
      042F 1953          ; This is a SHARED-DEFAULT user.
      042F 1954          ;
      50 0098 C5 9E 042F 1955 SHR_DEF:MOVAB UCBSQ_XQ_SHARE(R5),R0 ; Get address of SHR listhead
      51 50 D0 0434 1956          MOVL  R0,R1          ; Save address of start of listhead
      51 61 D0 0437 1957 10$: 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 12 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 12 0448 1964          SHR_G_DEST+4(R1)
      50 EB 12 044A 1965          BNEQ  10$          ; Br if not - check next in list
      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
      50 1A A3 3C 0453 1977          INCC  CDB_C_MBSCTR(R4) ; Count multicast blocks sent
      045D 1978          MOVZWL CXBSW_BCNT(R3),R0 ; Get BCNT
      0461 1979          CNTR  R0,CDB_L_MSNTCTR(R4),L ; Count multicast bytes sent
      046D 1980 20$:
      046D 1981          ;
      046D 1982          ; If we are running in point-to-point mode, then queue xmit on wait queue
      046D 1983          ; if we are waiting for run!
      046D 1984          ;
      00 91 046D 1985          CMPB  #NMASC_LINPR_POI,- ; % Are we in PT-TO-PT mode?
      00DC C5 046F 1986          UCBSB_XQ_PROTR5) ;
      OE 12 0472 1987          BNEQ  40$          ; % Br if not
      06 E1 0474 1988          BBC   #UCBSV_XQ_STACK,- ; % Br if not in stack wait state
      09 68 A5 0476 1989          UCBSW_DEVSTS(R5),40$ ;
      00B4 D5 63 OE 0479 1990          INSQUE (R3),UCBSQ_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

```



```

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$ALOUBAMAP ; Allocate UNIBUS map registers
OC 50 EB 04F6 2049 : BLBS R0,40$ ; Br if one available
OODC C4 63 OE 04F9 2050 : INSQUE (R3),CDB_Q_XMTREQ(R4) ; Re-insert CXB on request queue
50 01 9A 04FE 2051 20$: POPQ R6 ; Restore R6,R7
05 0504 2052 : MOVZBL S^#SS$_NORMAL,R0 ; Good return
0505 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
22 A3 57 90 0508 2061 50$: SETBIT R7,CDB_B_XMTMAP(R4) ; Set mapping slot in use flag
00000000'GF 16 0510 2062 : MOVB R7,CXB$B_XQ_SLOT(R3) ; Save mapping slot number used
0514 2063 : JSB G^IOC$LOADUBAMAPA ; Load map registers
051A 2064 :
051A 2065 : Find next ring entry and insert data
051A 2066 :
52 18 A4 9A 051A 2067 : MOVZBL CDB_B_NEXTXMT(R4),R2 ; Get next ring entry
18 A4 96 051E 2068 : INCB CDB_B_NEXTXMT(R4) ; Bump ring pointer
FC 8F 8A 0521 2069 : BICB #^C2MAX C XMT-1>,- ; Modulo xmit ring size
18 A4 0524 2070 : CDB_B_NEXTXMT(R4)
23 A3 52 90 0526 2071 : MOVB R2,CXB$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$W_BCNT(R3),R0 ; Get BYTE count
50 FF 8F 78 053C 2077 : INCL R0 ; Round up by one
50 50 053E 2081 : ASHL #-1,R0,R0 ; Convert to WORD count
06 A2 50 AE 0542 2082 : MNEGW R0,XMT_W_LEN(R2) ; Store message length (2's complement)
04 A2 7C A5 B0 0543 2083 : MOVW UCBSW_BOFF(R5),XMT_W_ADDR(R2) ; Move byte offset - BA0-BA8
34 A6 F0 054C 2084 : INSV CRB$L_INTD+VEC$W_MAPREG(R6),- ; Insert BA9-BA15
07 09 054F 2085 : #9,#7,XMT_W_ADDR(R2) ;&
04 A2 0551 2086 :
50 06 07 EF 0553 2086 : EXTZV #7,#6,CRB$L_INTD+VEC$W_MAPREG(R6),R0 ; Get BA16-BA21
50 34 A6 0556 2087 :
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 :
0560 .1 : Descriptor bit settings are calculated as follows.
0560 .2 :
0560 .3 : If the beginning address is Even or ODD, then the END address is:

```

:RNG0001
-4

:RNG0001
:RNG0001
:RNG0001

```

:RNG0001      0560      .4      :
:RNG0001      0560      .5      :
:RNG0001      0560      .6      :
:RNG0001      0560      .7      :
:RNG0001      0560      .8      :
:RNG0001      0560      .9      :
:RNG0001      0560     10      :
-8
50  A000 8F  B0 0560 2098      :
      11 7C A5  E9 0565 2100      :
      0569 2102      :
      0569 2103      :
      0569 2104      :
      11 1A A3  E8 0569 2105      :
      06 A6  B7 056D 2106      :
      08 11 0571 2107      :
      0575 2108      :
      0578 2109      :
      057A 2110      :
      057A 2111      :
      057A 2112      :
      04 1A A3  E9 057A 2113 70$:
      01 91 057E 2114      :
      20 A3 0582 2115 80$:
      04 13 0584 2116      :
      0A A6  B4 0586      .1      :
      0107 C4 96 0588      .2      :
      020A C4 04 90 058C      .3 90$:
      02 A6 50  A8 058F      .4      :
      0593      .5      :
      0598      .6      :
      059C      .7      :
      059C      .8      :
      059C      .9      :
      059C     10      :
52  0114 C4  D0 059F     11      :
      05A4     12      :
      05A4     17      :
      05A4     18      :
      05A4     19      :
      05A4     20      :
      0E A2 10  B3 05A4     25      :
      16 13 05A8     26      :
      51 23 A3 9A 05AA     28      :
      08 A2 51  D0 05AE     29      :
      51 F0 8F  B0 05B3     30      :
      51 78 05B7     31      :
      05BB     32      :
      0A A2 51 90 05BC     33 120$:
      00E8 D4 63 0E 05C0     34      :
      05C5     35      :
      05C5     36      :
      020A C4 04 90 05C5     37 130$:
      05CA     39      :
      FEBA 31 05CA     40      :
      :
      BEGINNING & LENTH, then END address is:
      -----
      Even Even Even
      Even Odd Odd
      Odd Even Odd
      Odd Odd Even
      :
      MOVW #XMT_DSC_M_VALID!- ; Build descriptor flag
      XMT_DSC_M_EOM,R0
      BLBC UCBSQ_BOFF(R5),70$ ; Br if even byte boundary at BEginning
      :
      Beginning buffer address is Odd
      :
      SETBIT #XMT_DSC_V_BEGODD,R0 ; Else, beginning is Odd
      BLBS CXBSQ_BCNT(R3),80$ ; Br if odd LENGTH, end is even
      SETBIT #XMT_DSC_V_ENDODD,R0 ; Else, ending is Odd
      DECW XMT_Q_LEN(R6) ; Increase length by 1 (complemented)
      BRB 80$ ; Continue
      :
      Beginning buffer address is Even
      :
      BLBC CXBSW_BCNT(R3),80$ ; Br if even LENGTH
      SETBIT #XMT_DSC_V_ENDODD,R0 ; Else, ending is Odd
      CMPB #XQ_FC_V_XMIT,- ; Is this a normal XMIT?
      CXBSB_XQ_FUNC(R3)
      BEQL 90$ ; Br if yes
      SETBIT #XMT_DSC_V_SETUP,R0 ; Else, indicate SETUP function
      CLRW XMT_Q_TDR(R6) ; Clear the TDR cell
      INCB CDB_B_XMTCNT(R4) ; Tally one more xmit in progress
      MOVW #XMT_TIM,CDB_B_TIM_XMT(R4) ; Start xmit timer
      BISW R0,XMT_W_ADDRHI(R6) ; Set descriptor bits
      :
      Poll the xmit ring to see if it went invalid
      :
      POPQ R6 ; Restore R6,R7
      MOVL CDB_L_CSR(R4),R2 ; Get CSR address
      :
      If the QNA has invalidated the TRANSMIT RING, then we must reset the
      starting address of the ring list to point to the current entry.
      :
      BITW #XQ_CSR_M_XMTINV,CSR(R2); Is transmit ring still valid?
      BEQL 120$ ; Br if yes
      MOVZBL CXBSB_XQ_RING(R3),R1 ; Else, get ring entry number
      MOVL CDB_L_XRINGPA(R4)[R1],R1; Get the buffer mapping value
      MOVW R1,XMTLIST(R2) ; Set address of transmit list entry
      ASHL #-16,R1,R1 ; Shift down high order address bits
      :
      MOVW R1,XMTLIST1(R2) ;
      INSQUE (R3),@CDB_Q_XMTPND+4(R4) ; Insert CXB on WAITING queue
      BITW #XQ_CSR_M_CAR,CSR(R2) ; Is carrier present?
      BEQL 130$ ; Br if not, okay - else
      INCC CDB_L_BIDCTR(R4) ; Count blocks sent - initially deferred
      MOVW #XMT_TIM,CDB_B_TIM_XMT(R4) ; Start xmit timer
      :
      BRW XMT_ALT_START ; Try for more requests

```


:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-26

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-4

```

05CD .41
05CD .42 XMT_UV1:
05CD .43 ; Transmit operation on MicroVAX I
05CD .44
05CD .45 ASSUME MAX_C_XMTUV1 LE 8
05CD .46 ASSUME MAX_C_XMTUV1 LT MAX_C_XMT ; Must not use all rings
05CD .47 FFC #0,#MAX_C_XMTUV1,CDB_B_XMTMAP(R4),R7 ; Find a free transmit slot
57 01 00 EB 05D0
18 A4 13 05D3 .48 BEQL 220$ ; Br if none free
7D 05D5 .49
05D5 .50 ; Move CXB data to contiguous buffer.
05D5 .51
53 00DC D4 OF 05D5 .52 REMQUE @CDB_Q_XMTREQ(R4),R3 ; Get oldest xmit request
76 1D 05DA .53 BVS 220$ ; Br if none
52 18 A3 3C 05DC 2143 MOVZWL CXBSW_BOFF(R3),R2 ; Get offset to start of data
52 52 53 CO 05E0 2144 ADDL R3,R2 ; Compute system buffer virtual address
51 00DB C447 DO 05E3 2145 MOVL CDB_L_XMT_VA(R4)[R7],R1 ; Get contiguous buffer's VA
38 BE 05E9 2146 PUSHR #^M<R3,R4,R5> ; Save registers
62 1A A3 28 05EB 2147 MOVCL CXBSW_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 MOVBL R2,CXBSW_XQ_RING(R3) ; Save ring entry number
22 A3 57 90 0602 2157 MOVBL R7,CXBSW_XQ_SLOT(R3) ; Save mapping slot number used
52 009C C442 DO 0606 2158 SETBIT R7,CDB_B_XMTMAP(R4) ; Set mapping slot in use flag
62 B4 060B 2159 MOVL CDB_L_XRINGVA(R4)[R2],R2 ; Get ring entry virtual address
8000 8F B0 0613 2160 CLRW XMT_W_FLAG(R2) ; Zero the FLAG word
08 A2 B0 0617 2161 MOVW #XMT_STS_M_LAST,- ; Init STATUS word
50 1A A3 3C 0619 2162 XMT_Q_STS(R2)
3C 50 B1 061D .1 MOVZWL CXBSW_BCNT(R3),R0 ; Get BYTE count
07 1E 0620 .2 CMPW R0,#MIN_PKT_SIZE+XQ_C_HEADER ; Is packet at least minimum size?
50 3C D0 0622 .3 BGEQU 210$ ; Br if yes, okay
0625 .4 MOVL #MIN_PKT_SIZE+XQ_C_HEADER,R0 ; Else set to minimum
0625 .5
0625 .6 ; *****
0625 .7 ; The following instruction is needed to keep the code honest, that
0625 .8 ; figures out whether the end byte is odd.
0625 .9 ; *****
1A A3 01 AA 0625 .10 BICW #1,CXBSW_BCNT(R3) ; Make the count look even!
50 FF 8F 78 0629 .11 210$: INCL R0 ; Round up by one
50 062B 2168 ASHL #-1,R0,R0 ; Convert to WORD count
062F
50 06  A2  50  AE 0630 2169 MNEGW R0,XMT_W_LEN(R2) ; Store message length (2's complement)
50 00C0 C447 DO 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
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
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-14

			064A	.1	
			064A	.2	
			064A	.3	
			064A	.4	
			064A	.5	
			064A	.6	
			064A	.7	
			064A	.8	
			064A	.9	
			064A	.10	
50	A000 8F	B0	064A	.12	
			064F	.13	
	FF28	31	064F	.14	
			0652	.15	
			0652	.16	220\$:
50	01	9A	0655	2190	
		05	0658	2191	
			0659	2192	
			0659	2193	
			0659	2194	

```

: Descriptor bit settings are calculated as follows.
: If the beginning address is Even or ODD, then the END address is:
: BEGINNING & LENTH, then END address is:
: -----
: Even      Even      Even
: Even      Odd       Odd
: Odd       Even      Odd
: Odd       Odd       Even
: Build descriptor flag
: Continue - always even start buffers!
: Restore R6,R7
: Good return
: Return to caller
.DSABL LSB

```

```

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 :
0659 2236 CLRW IRPSW BOFF(R3) ; Set no quota to here
0659 2237 MOVZBL S*#SS$ ACCVIO,R0 ; Assume access violation
0659 2238 MOVL P5(AP),R7 ; Get address for source address
0659 2239 BEQL 10$ ; Br if none
0659 2240 IFNOWRT #RHDR_C_DATA,(R7),20$ ; Check for write access to buffer
0659 2241 BBS #IRPSV DIAGBUF - ; Br if diagnostic buffer given
0659 2242 IRPSW_STS(R3),10$ ;
0659 2243 MOVZBL #RHDR_C_LENGTH,R1 ; Get size of header buffer
0659 2244 PUSHL R3 ; Save IRP address
0659 2245 JSB G^EXES$ALLOCBUF ; Allocate header buffer
0659 2246 POPL R3 ; Restore IRP address
0659 2247 BLBC R0,20$ ; Br if allocation failure
0659 2248 MOVL R2,IRPSL_DIAGBUF(R3) ; Save buffer address
0659 2249 BISW #IRPSM_DIAGBUF,IRPSW_STS(R3) ; Indicate diag buffer present
0659 2250 ASSUME RHDR_L_DATA EQ 0
0659 2251 MOVAB RHDR_T_DATA(R2),(R2)+ ; Set address of start of data

```

```

30 A3 B4
50 0C 9A
57 10 AC D0
33 13
07 E0
28 2A A3
51 1A 9A
53 DD
00000000 GF 16
53 BED0
40 50 E9
4C A3 52 D0
0080 8F AB
2A A3
82 0C A2 9E

```

```

      82  57  D0  068F  2252      ASSUME  RHDR_L_BUFFER EQ RHDR_L_DATA+4
      82  51  B0  068F  2253      MOVL   R7,(R2)+      ; Set user buffer address
      82  51  B0  0692  2254      ASSUME  RHDR_W_SIZE EQ RHDR_L_BUFFER+4
      82  13  90  0692  2255      MOVW   R1,(R2)+      ; Save size of allocation
      82  13  90  0695  2256      ASSUME  RHDR_B_TYPE EQ RHDR_W_SIZE+2
      50  14  9A  0695  2257      MOVB   #DYN$C_BUFIO,(R2)+ ; Set structure type
      51  04  AC  9A  0698  2258 10$: MOVZBL S^#SS$BADPARAM,R0 ; Assume illegal size
      51  04  AC  3C  0698  2259      MOVZWL P2(AP),R1      ; Get size
      069F  2260
      069F  2261
      069F  2262
      50  20  13  069F  2263      BEQL   20$          ; Br if zero - illegal
      3C  A3  50  D0  06A1  2264      MOVL   P1(AP),R0    ; Get user buffer address
      00000000'GF 16 06A4  2265      MOVL   R0,IRP$L_X0 DATBUF(R3) ; Save user VA for completion
      2A  A3  20  A8  06A8  2266      JSB   G^EXE$READCRK ; Check the buffer
      06AE  2267      ; (No return on NO ACCESS)
      06AE  2268      BISW  #IRPSM_CHAINED,IRPSW_STS(R3) ; Allow data chaining
      06B2  2269      SETIPL UCBSB_FIPL(R5) ; Raise IPL to lock data base
      06  12  10  06B6  2270      BSBB  RCV_START    ; Process the request
      00000000'GF 06 50 E9 06B8  2271      BLBC  R0,20$      ; Br if error
      06C1  2272      JMP   G^EXE$QIORETURN ; Else, take normal return
      FC7C  31  06C1  2274 20$: 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:
06C4 2303      :     MOVZWL #SS$_DEVINACT,R0      ; Setup return status
06C9 2304      :     RSB                          ; Return to caller
06CA 2305      :
06CA 2306      :     .ENABL LSB
06CA 2307      : RCV_START::
06CA 2308      :     BBC #UCB$_XQ_RUN,-          ; Start receive I/O operation
06CC 2309      :     UCBSW_DEVSTS(R5),INACT_ERROR ; Br if UCB is not in RUN mode
06CF 2310      :
06CF 2311      :     MOVL UCBS$_XQ_CDB(R5),R4    ; Get CDB address
06D4 2313      :     BEQL 40$                    ; Br if none
06D6 2314      :     BBC #CDB_STS_V_RUN,-       ; Br if QNA not running
06D8 2315      :     CDB_B_STS(R4),INACT_ERROR
06DC 2316      :     MOVAB UCBSQ_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      :
06E1 2321      :     BBC #UCB$_XQ_SHARE,-       ; Br if UCB is NOT SHARED
06E3 2322      :     UCBSW_DEVSTS(R5),5$      ;
06E6 2323      :
06E6 2324      :     : Try to find a match on PID/CHAN
06E6 2325      :
06E6 2326      :     BSBW MATCH_SHR            ; Try to find shared user
06E9 2327      :     BNEQ INACT_ERROR          ; Br if none - inactive user
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      :
06EF 2332      :     5$: REMQUE @ (R1)+,R2      ; Dequeue a received message
06F2 2333      :     ; ..bump pointer to end of list pointer

```

```

50 20D4 BF 3C
05
04 E1
F5 68 A5
54 00C8 C5 D0
39 13
01 E1
EB 0246 C4
51 00A0 C5 9E
03 E1
09 68 A5
0027 30
D9 12
51 18 A1 9E
91 0F
52 91 0F

```

```

:RNG0001
-2

```

```
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 arrival unless IOSM_NOW specified
      06F9 2342      :
OA 20 A3 06 E1 06F9 2343 10$: BBC      #IOSV NOW,IRPSW FUNC(R3),20$ ; Br if not READ NOW
50    50 0870 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   UCBSQ_XQ_RCVREQ EQ UCBSQ_XQ_RCVMSG+8
      0708 2351      ASSUME   SHR_Q_RCVREQ EQ SHR_Q_RCVMSG+8
08 B1 63 0E 0708 2352 20$: INSQUE  (R3),R8(R1) ; Put packet on waiting list
      50 01 9A 070C 2353 30$: MOVZBL  S^#SS$_NORMAL,R0 ; Set QIO status return
      05 070F 2354 40$: 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:
51  00C4 C5  D0 0710 2373      : Try to find shared user
                    :     MOVL   UCBSL_XQ_DEFUSR(R5),R1      : Get address of default user
                    :     BEQL   10$      : Br if no default user
                    :     BSBB  CHECK_SHR      : Check for match
50  0098 C5  9E 0719 2376      :     BEQL   40$      : Br if match
                    :     MOVAB  UCBSQ_XQ_SHARE(R5),R0      : Save address of listhead
                    :     MOVL   R0,R1      : Copy listhead address
                    :     ASSUME  SHR_L_QFL EQ 0
51  61  D0 0723 2380 20$:      :     MOVL   (R1),R1      : Get next in list
50  51  D1 0726 2381      :     CMPL   R1,R0      : Back to start of list?
                    :     BEQL   30$      : Br if yes - no pid/chan match
                    :     BSBB  CHECK_SHR      : Check for match
                    :     BNEQ  20$      : Br if none
                    :     BRB   40$      : Return in success
50  50  D0 0731 2386 30$:      :     MOVL   R0,R0      : Return match failure
                    :     RSB
                    :
0735 2388      :
0735 2389      :+
0735 2390      : Subroutine to check if PID and SHR data base match up
0735 2391      :
0735 2392      : Inputs:
0735 2393      :     R1 = Address of SHR
0735 2394      :     R3 = Address of IRP
0735 2395      :
0735 2396      : Outputs:
0735 2397      :     Z-Bit set then match.
0735 2398      :     Z-Bit clear then no match.
0735 2399      :-
0735 2400      :
0735 2401 CHECK_SHR:
                    :     TSTL   IRP$L_PID(R3)      : Check for match with SHR data base
0C  A3  D5 0735 2402      :     BLSS  10$      : Is this an internal IRP user?
                    :     BR   10$      : Br if yes, only one allowed per UCB
                    :
073A 2404      : Normal QIO user
073A 2405      :
0C  A1  OC  A3  D1 073A 2407      :     CMPL   IRP$L_PID(R3),SHR_L_PID(R1) ; PIDs match?
                    :     BNEQ  30$      : Br if no - try for next
                    :     BRB   20$      : Else, continue checks
                    :
0743 2410      : Internal IRP user
0743 2411      :
00B8 C5  D1 0743 2413 10$:      :     CMPL   UCBSL_XQ_PID(R5),SHR_L_PID(R1) ; Is this the Internal user?

```

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



```

0751 2419 .SBTTL ALT_START - ALTERNATE START I/O ROUTINE
0751 2420 :++
0751 2421 : ALT_START - ALTERNATE START 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 : IRPSL_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::
0751 2451 BBS #IRPSV_FUNC,- ; Accept an "internal" IRP
0751 2452 IRPSW_STS(R3),10$ ; If BS then read function
0751 2453 MOVW UCBSW_XQ_PROTYP(R5),- ; MUST be a non-promiscuous user
0751 2454 IRPSW_XQ_PROTYP(R3) ;
0751 2455 MOVL IRPSL_SVAPTE(R3),R2 ; Get address of start of data
0751 2456 SUBL3 #XQ_C_HEADER,(R2),R1 ; Get the xmit buffer address
0751 2457 SUBL R2,R1 ; Form offset to start of data
0751 2458 MOVW R1,CXBSW_BOFF(R2) ; Store offset in CXB
0751 2459 MOVW #DYN$C_CXB,CXBSB_TYPE(R2) ; Set structure type to CXB
0751 2460 CMPB UCBSB_XQ_PRO(R5),- ; % Point-to-point mode?
0751 2461 #NMASC_LINPR_POI ;
0751 2462 BNEQ 5$ ; % If so,
0751 2463 CLRQ IRPSQ_STATION(R3) ; % Pick up destination from SHR block
0751 2464 5$: PUSHL R3 ; Save IRP address
0751 2465 BSBW XMT_START ; Start transmit operation
0751 2466 POPL R3 ; Restore IRP address
0751 2467 BRB 30$ ; Continue
0751 2468
0751 2469 10$: MOVL IRPSL_SVAPTE(R3),R2 ; Get address of input buffer
0751 2470 BEQL 20$ ; Br if none
0751 2471 MOVL UCBSL_XQ_CDB(R5),R4 ; Get CDB address
0751 2472 CLRL IRPSL_SVAPTE(R3) ; Make sure SVAPTE is cleared
0751 2473
0751 2474 :
0751 2475 : The driver must be prepared to process chained buffers returned from
0751 2476 : the higher levels.

```

```

01 E0
2D 2A A3
00CE C5 B0
3A A3
52 2C A3 D0
51 62 0E C3
51 51 52 C2
18 A2 51 B0
0A A2 1B 90
00DC C5 91
00
03 12
40 A3 7C
53 DD
FCOB 30
53 BED0
14 11
52 2C A3 D0
0B 13
54 00CB C5 D0
2C A3 D4

```

		0791	2477	:						
		0791	2478	:&&15\$:	PUSHL	CXBSL_LINK(R2)		:	Save address of next in link	
		0791	2479	:&&	CLRL	CXBSL_LINK(R2)		:	Clear the link cell	
OCA1	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 -- IOSM_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 IRPSW 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 -- IOSM_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 -- IOSM_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 ADDRESS
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::
2C A3 D4 079E 2554 CLR    IRPSL_SVAPTE(R3) ; SET MODE FDT processing
38 A3 7C 07A1 2555 CLR    IRPSL_MEDIA(R3) ; Set no buffered packet
0094 C3 D4 07A4 2556 CLR    IRPSL_XQ_SETUP(R3) ; Reset mode data area
50 0084 8F 3C 07A8 2557 MOVZWL #SSS_DEVOFFLINE,R0 ; Indicate no SETUP buffer yet
04 E0 07AD 2558 BBS    #UCBSV_ONLINE,- ; Assume unit if offline
03 64 A5 07AF 2559 UCBSW STS(R5),5$ ; Br if unit online
FB8B 31 07B2 2560 BRW    ABORTIO ; Else, abort the I/O request
07B5 2561
57 20 A3 B0 07B5 2562 5$: MOVW  IRPSW_FUNC(R3),R7 ; Get entire function code
5C 57 08 E1 07B9 2563 BBC    #IOSV_ATTNAST,R7,30$ ; Br if not attention AST
07BD 2564 :
07BD 2565 : User is requesting an attention AST
07BD 2566 :
57 00C0 C5 DE 07BD 2567 MOVAL  UCBSL_XQ_AST(R5),R7 ; Get address of AST list
00000000*GF 16 07C2 2568 JSB    G*COMBSETATTNAST ; Set up attention AST
07C8 2569
12 68 A5 E9 07C8 2570 ASSUME UCBSV_XQ_INITED EQ 0
07C8 2571 BLBC  UCBSW_DEVSTS(R5),0$ ; Br if protocol not active
07CC 2572
51 00A0 C5 9E 07CC 2573 MOVAB  UCBSQ_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 D0 07D6 2576 PUSHL R3 ; Save IRP address
228B 30 07D8 2577 BSBW  POKE_USER ; Inform user
53 B E D 0 07DB 2578 POPL  R3 ; Restore IRP address
50 01 9A 07DE 2579 10$: MOVZBL S*#SSS_NORMAL,R0 ; Set success
51 44 A5 D0 07E1 2580 MOVL  UCBSL_DEVDEPEND(R5),R1 ; Get device dependent information
00000000*GF 17 07E5 2581 20$: JMP   G*EXESFINISHIO ; 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    #IOSV_SHUTDOWN,R7,27$ ; Br if not a shutdown request
07EF 2588 SETIPL UCBSB_FIPL(R5) ; Sync access to UCB & CDB
07F3 2589 CLRBIT #UCBSV_XQ_RUN,- ; Clear the RUN flag
07F3 2590 UCBSW_DEVSTS(R5)
52 28 A3 3C 07F8 2591 MOVZWL IRPSW_CHAN(R3),R2 ; Get channel number
07FC 2592 ASSUME CANSC_CANCEL EQ 0
58 D4 07FC 2593 CLR    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$: BBC    #IOSV_STARTUP,R7,10$ ; Br if not a startup request
0807 2598 CLRBIT #XMSV_ERR_START,UCBSL_DEVDEPEND(R5) ; Clear start error flag
0060 8F AB 080C 2599 BISW  #UCBSM_XQ_START!UCBSM_XQ_STACK,- ; Set start and stack states
68 A5 0810 2600 UCBSW_DEVSTS(R5) ;
0812 2601 SETBIT #UCBSV_XQ_RUN,- ; Set the RUN flag
0812 2602 UCBSW_DEVSTS(R5)
C5 11 0817 2603 BRB    10$ ; Complete the request

```

:RNG0001
-2

```

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$: TSTL UCBSL_XQ_CDB(R5) : Is CDB there?
-2 OE 12 081D 2610 BNEQ 35$ : Br if yes
1691 30 081F 2611 BSBW ALLOC_CDB : Else, allocate CDB
08 50 E8 0822 2612 BLBS RO,35$ : Br if successful
50 0124 8F 3C 0825 2613 MOVZWL #SS$ INSMEM,RO : 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 RO,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 #IOSV_CTRL,R7,36$ : Br if line request
52 F8BE CF 9E 083C 2620 MOVAB CIRCUIT_PARAM,R2 : Else, use the circuit parameters
1D05 30 0841 2621 36$: BSBW VALIDATE_P2 : Validate the P2 parameters
9E 50 E9 0844 2622 BLBC RO,20$ : Br if error
A0 57 09 E1 0847 2623 BBC #IOSV_CTRL,R7,25$ : Br if not a LINE request
084B 2624 SETIPL UCBSB_FIPL(R5) : Sync access to UCB's
1CCB 30 084F 2625 BSBW SAV_MULTI : Save the multicast address list
0090 C3 D4 0852 2626 CLRL IRPSL_XQ_SHR(R3) : ; Assume exclusive user
03 E1 0856 2627 BBC #UCBSV_XQ_SHARE,- : Br if not a SHARED user
16 68 A5 0858 2628 UCBSW_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,IRPSL_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 UCBSG_XQ_DES(R5) : address in the UCB
16 A1 B0 086B 2638 MOVL SHR_G_DEST+4(R1),- :
00D4 C5 086E 2639 UCBSG_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$: MOVL UCBSL_XQ_CDB(R5),R1 : Get CDB address
-2 13 0246 C1 E9 0876 2647 ASSUME CDB_STS_V_INITED EQ 0
087B 2648 BLBC CDB_B_STSTR1),45$ : Br if controller not enabled, use
51 0249 C1 9E 087B 2649 : the fixed defaults
52 00E1 C5 9E 0880 2650 MOVAB CDB_B_SETPRM(R1),R1 : Get address of settable parameters
50 01 9A 0885 2651 MOVAB UCBSB_XQ_CDBPRM(R5),R2 : Get address of UCB parameters
82 81 90 0888 2652 MOVZBL #UCBSV_XQ_CDBPRM,RO : Get size of parameters to move
FA 50 F5 088B 2653 42$: MOVB (R1)+(R2)+ : Store CDB parameters into UCB
52 F7E6 CF 9E 088E 2654 SOBGTR RO,42$ : Loop if more
1DEE 30 0893 2655 45$: MOVAB LINE_PARAM_WO,R2 : Get address of verification table
0090 C3 D0 0896 2656 BSBW CHANGE_PARAM : Change the parameters
0C 13 089B 2657 MOVL IRPSL_XQ_SHR(R3),R1 : Get the SHR structure address
00D0 C5 D0 089D 2658 BEQL 50$ : Br if not present, skip it
12 A1 08A1 2659 MOVL UCBSG_XQ_DES(R5),- : Else, reset the destination user
00D4 C5 B0 08A3 2660 MOVL SHR_G_DEST(R1) : address into the SHR structure
16 A1 08A7 2661 MOVL UCBSG_XQ_DES+4(R5),- :
2662 SHR_G_DEST+4(R1) :

```

:RNG0001
-1

```

0E 57 07 E1 08A9 2663
08A9 2664 50$: BBC #IOSV_SHUTDOWN,R7,60$ ; Br if not shutdown request
08AD 2665 :
08AD 2666 : Shutdown protocol request
08AD 2667 :
08AD 2668 ASSUME UCBSV_XQ_INITED EQ 0
03 68 A5 E8 08AD 2669 BLBS UCBSW_DEVSTS(R5),55$ ; Br if still inited
FF2A 31 08B1 2670 BRW 10$ ; Else, complete I/O request now
21 A3 03 90 08B4 2671 55$: MOVB #XQ_FC_V_STOP,IRPSB_XQ_FUNC(R3) ; Set internal function code
03F8 31 08B8 2672 BRW QUEPKT ; Queue request to QNA
08BB 2673
03 57 06 E0 08BB 2674 60$: BBS #IOSV_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$: MOVB #XQ_FC_V_INIT,IRPSB_XQ_FUNC(R3) ; Insert internal function code
08C6 2680 :
08C6 2681 : If the UCB is already initialized for SHARED use, then we will
08C6 2682 : check to make sure that the SHR structure exists. If it does then
08C6 2683 : the share structure must be active, by definition.
08C6 2684 :
0090 C3 D5 08C6 2685 TSTL IRPSL_XQ_SHR(R3) ; Was the SHR structure present
F3 12 08CA 2686 BNEQ 70$ ; Br if yes, already started
03 E0 08CC 2687 BBS #UCBSV_XQ_SHARE,- ; Br if SHARED UCB, ignore status
61 68 A5 08CE 2688 UCBSW_DEVSTS(R5),125$ ; ..make user a shared user
EA 68 A5 E8 08D1 2689 ASSUME UCBSV_XQ_INITED EQ 0
08D1 2690 BLBS UCBSW_DEVSTS(R5),70$ ; Br if already started
08D5 2691 : ..check multicast address list
43 68 A5 E1 08D5 2692 BBC #UCBSV_XQ_PROTYP,- ; Br if no protocol specified
08D7 2693 UCBSW_DEVSTS(R5),100$ ; ..error
08DA 2694 :
08DA 2695 : Check if protocol type is to be shared
08DA 2696 :
00D8 03 91 08DA 2697 CMPB #NMASC_ACC_EXC,- ; Is this PROTOCOL TYPE for exclusive
C5 08DC 2698 UCBSB_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 UCBSW_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 NMASC_STATE_ON EQ 0
08ED 2713 ASSUME NMASC_STATE_OFF EQ 1
08ED 2714 :
1B 00DE C5 E8 08ED 2715 BLBS UCBSB_XQ_PRM(R5),90$ ; Br if NOT a PROMISCUOUS user
08F2 2716 :
08F2 2717 : The promiscuous user must have PHY_IO privilege
08F2 2718 :

```

:RNG0001
-2

```

00002ABD'EF 9E 08F2 2719      MOVAB  MATCH_PROMPTYP,R0      ; Get address of Action routine
   50          08F8
   1C31        30 08F9 2720      IFPRIV PHY_IO,90$             ; If user has privilege, then okay
   50 24       3C 08FF 2721      BSBW   RES_MULTI              ; Else, Restore original multicast list
51 0B18 8F     3C 0902 2722      MOVZWL #SS$ NOPRIV,R0        ; Return error - NOPRIV
   FED8       3C 0905 2723      MOVZWL #NMASC_PCLI_PRM,R1    ; Return the bad parameter
   31         090A 2724      BRW    20$                    ; Finish the I/O request
54 00C8 C5     D0 090D 2725 90$:  PUSHQ  R4                    ; Save PCB, UCB addresses
   60         0910 2726      MOVL   UCB$L_XQ_CDB(R5),R4   ; Get CDB address
   22 50       E9 0915 2728      JSB    (R0)                  ; Try to find exact match
   22 50       E9 0917 2729      POPQ   R4                    ; Restore PCB, UCB addresses
   22 50       E9 091A 2730      BLBC   R0,130$              ; Br if none found - okay
   22 50       E9 091D 2731      ;                               ;
   22 50       E9 091D 2732      ;                               ; Bad protocol type
   22 50       E9 091D 2733      ;                               ;
51 0B0E 8F     3C 091D 2734 100$: MOVZWL #NMASC_PCLI_PTY,R1    ; Return bad parameter code
   05         11 0922 2735      BRB    120$                  ; Finish error reporting
   05         11 0924 2736      ;                               ;
   05         11 0924 2737      ;                               ; Bad quota calculated
   05         11 0924 2738      ;                               ;
51 0451 8F     3C 0924 2739 110$: MOVZWL #NMASC_PCLI_BFN,R1    ; Return bad parameter code
   50 14       9A 0929 2740 120$: 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
   FEB3       31 0932 2743      ;                               ;
   FEB3       31 0932 2744      ;                               ; Shared protocol type - look for same protocol type in other UCB.
   FEB3       31 0932 2745      ;                               ;
   0139       30 0932 2746 125$: BSBW   SHR_UCB                    ; Find other UCB in user or make this
   F4 50       E9 0935 2747      ;                               ; UCB shareable
   F4 50       E9 0935 2748      BLBC   R0,123$              ; Br on error
   F4 50       E9 0938 2749      ;                               ;
   F4 50       E9 0938 2750      ;                               ; We must now check if UCB is already inited.
   F4 50       E9 0938 2751      ;                               ;
   F4 50       E9 0938 2752      ;                               ; ASSUME UCB$V_XQ_INITED EQ 0
   F4 50       E9 0938 2753      BLBC   UCB$W_DEVSTS(R5),135$ ; Br if this UCB is NOT inited
   F4 50       E9 093C 2754      ;                               ; skip quota taking, already done
   008C       31 093C 2755      BRW    185$                  ; Else, compute multicast list
   008C       31 093F 2756      ;                               ;
   008C       31 093F 2757 130$: SETIPL #IPL$_ASTDEL            ; Reset IPL to ASTDEL
   008C       31 0942 2758      ;                               ;
   008C       31 0942 2759      ;                               ; Take quota needed
   008C       31 0942 2760      ;                               ;
   0097       30 0942 2761      BSBW   TAKE_QUOTA            ; Take quota from user
   DC 50       E9 0945 2762      BLBC   R0,1T0$              ; Br if error
00CC C5 57     B0 0948 2763      MOVW   R7,UCB$W_XQ_QUOTA(R5) ; Save quota in UCB
   00CC C5 57     B0 094D 2764 135$: SETBIT #UCB$V_XQ_INITED,-    ; Indicate unit is initialized
   00CC C5 57     B0 094D 2765      ;                               ; UCB$W_DEVSTS(R5)
   00CC C5 57     B0 0952 2766      ;                               ; SETIPL UCB$B_FIPL(R5)
   00CC C5 57     B0 0952 2766      ;                               ; Sync access to UCB and CDB
   0A 00DE C5   E8 0956 2767      BLBS   UCB$B_XQ_PRM(R5),140$ ; Br if NOT a PROMISCUOUS user
54 00C8 C5     D0 095B 2767      MOVL   UCB$L_XQ_CDB(R5),R4   ; Get CDB address
0210 C4 55     D0 0960 2770      MOVL   R5,CDB_L_PRMUSER(R4) ; Store promiscuous user's address
   0210 C4 55     D0 0965 2771      ;                               ;
   0210 C4 55     D0 0965 2772      ;                               ; Pre-allocate all needed receive buffers, if the CDB is not initialized yet!
   0210 C4 55     D0 0965 2773      ;                               ; The buffers are immediately deallocated, but this pre-allocation will allow
   0210 C4 55     D0 0965 2774      ;                               ; the pool to grow if necessary! This must be done here, before we run on the
   0210 C4 55     D0 0965 2775      ;                               ; interrupt stack.
   0210 C4 55     D0 0965 2776      ;                               ;

```

:RNG0001
-2

```

51 00DA C5 3C 0965 2777 140$: PUSHQ R3 ; Save R3, R4
54 00D6 C5 3C 0968 2778 MOVZWL UCBSW_XQ_BSZ(R5),R1 ; Get size of receive buffer
54 54 51 C6 096D 2779 MOVZWL UCBSW_XQ_HBQ(R5),R4 ; Get device buffer quota
54 54 D6 0972 2780 DIVL R1,R4 ; Compute number of buffers to allocate
7E D4 0975 2781 INCL R4 ; Plus one extra
00000000'GF 16 0977 2782 CLRL -(SP) ; End of list marker
OE 50 E9 0979 2783 145$: JSB G^EXE$ALONONPAGED ; Allocate the memory
08 A2 51 B0 097F 2784 BLBC R0,150$ ; Br on error
OA A2 1B 90 0982 2785 MOVW R1,IRPSW_SIZE(R2) ; Save size of buffer
7E 52 D0 0986 2786 MOVB #DYN$C(CXB,IRPSB_TYPE(R2)) ; Set structure type
E9 54 F5 098A 2787 MOVL R2,-(SP) ; Save buffer address
0990 2788 SOBGTR R4,145$ ; Loop if more to allocate
50 8E D0 0990 2789
00000000'GF 16 0993 2790 150$: MOVL (SP)+,R0 ; Get buffer address
08 13 0993 2791 BEQL 155$ ; Br if end of list
F3 11 0995 2792 JSB G^EXE$DEANONPAGED ; Deallocate the block
0998 2793 BRB 150$ ; Try for more
099D 2794 155$: POPQ R3 ; Restore R3, R4
09A0 2795
1AF0 30 09A0 2796 BSBW ADD MULTI ; Compile a new multicast address list
33 50 E8 09A3 2797 BLBS R0,T90$ ; Br if all okay
09A6 2798
09A6 2799 ASSUME NMASC_STATE_ON EQ 0
09A6 2800 ASSUME NMASC_STATE_OFF EQ 1
09A6 2801
04 00DE C5 E8 09A6 2802 BLBS UCBSB_XQ_PRM(R5),160$ ; Br if NOT a PROMISCUOUS user
0210 C4 D4 09AB 2803 CLRL CDB_L_PRMUSER(R4) ; Else, clear the PROMISCUOUS user addr
09AF 2804 160$: CLRBIT #UCBSV_XQ_INITED,- ; Indicate unit is not initialized
09AF 2805 UCBSW_DEVSTS(R5)
20 A6 57 C0 09B4 2806 ADDL R7,JIB$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 #NMASC_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$: ASSUME UCBSV_XQ_INITED EQ 0
03 68 A5 E8 09C4 2814 BLBS UCBSW_DEVSTS(R5),185$ ; Br if unit initied
FE13 31 09C8 2815 BRW 10$ ; Else, success
09CB 2816
09CB 2817 185$: SETIPL UCBSB_FIPL(R5) ; Sync access to UCB and CDB
1AC1 30 09CF 2818 BSBW ADD MULTI ; Compile global multicast address list
E7 50 E9 09D2 2819 BLBC R0,T70$ ; Br if error
21 A3 06 90 09D5 2820 MOVB #XQ_FC_V_CHMODE,IRPSB_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:
09DC 2848 : CLRL R0 : Assume failure
09DE 2849 : INCW UCBSW_XQ_BSZ(R5) : Round buffer size to even value
09E2 2850 : BICW #1,UCBSW_XQ_BSZ(R5) :
09E7 2851 : MOVZBL UCBSB_XQ_BFN(R5),R1 : Get number of receive buffers
09EC 2852 : MOVZWL UCBSW_DEVBUFSIZ(R5),R2 : Get buffer size
09F0 2853 : MULL R2,R1 : Get needed quota
09F3 2854 : MOVZWL R1,R7 : Copy quota
09F6 2855 : CMPL R1,R7 : Overflow?
09F9 2856 : BNEQ 110$ : Br if yes - error
09FB 2857 : MOVL PCB$L_STS(R4),-(SP) : Save current PCB status
09FF 2858 : SETBIT #PCBSV_SSRWAIT,- : Do not go into a resource wait
09FF 2859 : PCB$L_STS(R4) : just to check the quota
0A04 2860 : PUSHL R3 : Save R3
0A06 2861 : JSB G^EXE$BUFQUOPRC : Check quota
0A0C 2862 : POPL R3 : Restore R3
0A0F 2863 : MOVL (SP)+,PCB$L_STS(R4) : Restore previous PCB status
0A13 2864 : BLBS R0,50$ : Br if success
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 50 14 9A 0A28 2871 80$: CLRL R1 : No error return
0A2A 2872 90$: RSB
0A2B 2873 :
51 50 14 9A 0A2B 2874 110$: MOVZBL #SS$ BADPARAM,R0 : Setup error code
0A2E 2875 : MOVZWL #NMA$C_PCLI_BFN,R1 : Assume BAD BFN
0A33 2876 : BRB 90$ : 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 55 DD 0A38 2898     PUSHL   R5           : Save UCB address
55 57 DO 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 DO 0A45 2903     MOVL    R7,R9         : Copy quota taken
51 0190 C5 3C 0A48 2904     MOVZWL UCB$W_XQ_TOTQUO(R5),R1 : Get total quota
51 59 C0 0A4D 2905     ADDL    R9,R1         : Compute new total
50 51 3C 0A50 2906     MOVZWL R1,R0         : Copy quota
50 51 D1 0A53 2907     CML    R1,R0         : Overflow?
50 07 12 0A56 2908     BNEQ   110$         : Br if yes, error
50 01 9A 0A58 2909     MOVZBL #1,R0        : Else, return success
0A5B 2910 90$:     POPQ    R6           : Restore R6, R7
0A5E 2911     RSB           : Return to caller
0A5F 2912
20 A6 59 C0 0A5F 2913 110$:     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 #$$$_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 2953 SHR_UCB::
0A6E 2954     PUSHR  #M<R8,R9>          ; Setup shared protocol UCB
0A6E 2955     JSB   G^SCH$IOLOCKW      ; Save registers
0A6E 2956     MOVL  R5,R7              ; Lock I/O data base for write access
0A6E 2957     MOVZWL #SS$ DUPUNIT,R0   ; Save UCB address
0A6E 2958     CMPW  #1,UCB$W_REF(R5) ; Assume 2 channels assigned to UCB
0A6E 2959     BNEQ  23$              ; Is there only one reference to UCB?
0A6E 2960     MOVZBL UCB$B_XQ_ACC(R5),R8 ; Br if not, error (more than 1 channel)
0A6E 2961     BBS   #UCB$V_XQ_SHARE,- ; Save access mode
0A6E 2962     UCBSW_DEVSTS(R5),10$    ; Br if we are already the SHARED UCB
0A6E 2963     PUSHL  R4              ; Save PCB address
0A6E 2964     MOVL  UCB$SL_XQ_CDB(R5),R4 ; Get CDB address
0A6E 2965     MOVW  UCB$W_XQ_PROTYP(R5),R1 ; Get protocol type
0A6E 2966     BSBW  MATCH_PROTYP      ; Try to match protocol type
0A6E 2967     POPL  R4              ; Restore PCB address
0A6E 2968     BLBS  R0,10$         ; Br if found
0A6E 2969     MOVL  R7,R5          ; Else, get back old UCB address
0A6E 2970     CLRL  UCB$SL_PID(R5) ; Make this UCB shareable
0A6E 2971     MOVL  UCB$SL_ORB(R5),R0 ; Get the ORB address
0A6E 2972     CLRL  ORB$SL_OWNER(R0) ; clear owner UIC as well
0A6E 2973     CLRL  UCB$SL_XQ_DEFUSR(R5) ; Clear default user
0A6E 2974     BISW  #UCB$M_XQ_SHARE,- ; Indicate that UCB is in SHARED mode
0A6E 2975     UCBSW_DEVSTS(R5)
0A6E 2976     ;
0A6E 2977     ;

```

:RNG00C1
-2

```

OAB9 2978 : Allocate a share data structure and link it in
OAB9 2979 :
58 02 91 OAB9 2980 10$: CMPB #NMASC_ACC_LIM,R8 : Is the new user a limited user?
OB 13 OABC 2981 BEQL 20$ : Br if yes - okay
50 0840 8F 3C OABE 2982 MOVZWL #SS$ DEVALLOC,R0 : Assume protocol already allocated
00C4 C5 D5 OAC3 2983 TSTL UCBSXQ_DEFUSR(R5) : Is there already a default user?
2F 12 OAC7 2984 BNEQ 23$ : Br if YES - error
00C8 30 OAC9 2985 20$: BSBW CHECK_PARAM : Check out all parameters
2B 50 E9 OACC 2986 BLBC R0,25$ : Br on error
FF63 30 OACF 2987 BSBW CHECK_QUOTA : Check our quota
25 50 E9 OAD2 2988 BLBC R0,25$ : Br if error
51 2A 9A OAD5 2989 MOVZBL #SHR_C_LENGTH,R1 : Get size of structure to allocate
53 DD OAD8 2990 PUSHL R3 : Save IRP address
00000000 GF 16 OADA 2991 JSB G*EXESALLOCBUF : Allocate buffer, reset IPL to ASTDEL
53 8ED0 OAE0 2992 POPL R3 : Restore IRP address
1A 50 E8 OAE3 2993 BLBS R0,30$ : Br if success
50 0124 8F 3C OAE6 2994 MOVZWL #SS$ INSMEM,R0 : Else, return error reason
51 0080 C4 D0 OAEB 2995 MOVL PCB$JIB(R4),R1 : Get JIB address
20 A1 59 C0 OAF0 2996 ADDL R9,JIB$SL_BYTCNT(R1) : Restore quota
24 A1 59 C0 OAF4 2997 ADDL R9,JIB$SL_BYTLM(R1) : ..and byte limit
51 D4 OAF8 2998 23$: CLRL R1 : No bad parameter code
55 57 D0 OAF9 2999 25$: MOVL R7,R5 : Get back the OLD UCB address
0085 31 OAFD 3000 BRW 80$ : Exit with error
OAB0 3001 :
OAB0 3002 : Initialize shared (SHR) data structure
OAB0 3003 :
OAB0 3004 30$: ASSUME SHR_L_QFL EQ 0
OAB0 3005 ASSUME SHR_L_QBL EQ SHR_L_QFL+4
82 7C OAB0 3006 CLRQ (R2)+ : Zero LINK pointers
OAB0 3007 ASSUME SHR_W_SIZE EQ SHR_L_QBL+4
82 51 B0 OAB0 3008 MOVW R1,(R2)+ : Save size
OAB0 3009 ASSUME SHR_B_TYPE EQ SHR_W_SIZE+2
52 D6 OAB0 3010 INCL R2 : Filled by EXESALLOCBUF routine
OAB0 3011 ASSUME SHR_B_STS EQ SHR_B_TYPE+1
82 01 90 OAB0 3012 MOVB #SHR_STS_M_INITED,(R2)+ : Initialize SHR status
OAB0 3013 ASSUME SHR_C_PID EQ SHR_B_STS+1
82 0C A3 D0 OAB0 3014 MOVL IRP$PID(R3),(R2)+ : Save users PID and CHAN for
OAB0 3015 ASSUME SHR_W_CHAN EQ SHR_L_PID+4
82 28 A3 B0 OAB0 3016 MOVW IRP$W_CHAN(R3),(R2)+ : for future lookups
OAB0 3017 ASSUME SHR_G_DEST EQ SHR_W_CHAN+2
82 00D0 C7 D0 OAB0 3018 MOVL UCBSGXQ_DES(R7),(R2)+ : Save destination address
82 00D4 C7 B0 OAB0 3019 MOVW UCBSGXQ_DES+4(R7),(R2)+
OAB0 3020 ASSUME SHR_Q_QUEUES EQ SHR_G_DEST+6
51 02 9A OAB0 3021 MOVZBL #SHR_C_QUEUES,R1 : Get number of queues in structure
82 82 62 DE OAB0 3022 40$: MOVAL (R2)-(R2)+ : Set forward link pointer
82 FC A2 D0 OAB0 3023 MOVL -4(R2),(R2)+ : Set backward link pointer
F6 51 F5 OAB0 3024 SOBGTR R1,40$ : Loop if more listheads
OAB0 3025 ASSUME SHR_W_QUOTA EQ SHR_Q_QUEUES+<8*SHR_C_QUEUES>
82 59 B0 OAB0 3026 MOVW R9,(R2)+ : Initialize quota
OAB0 3027
OAB0 3028 ASSUME UCBSV_XQ_INITED EQ 0
00C 68 A5 E9 OAB0 3029 BLBC UCBSW_DEVSTS(R5),50$ : Br if UCB not initialized
00CC C5 59 A0 OAB0 3030 ADDW R9,UCBSW_XQ_QUOTA(R5) : Add to the current quota
0190 C5 59 A0 OAB0 3031 ADDW R9,UCBSW_XQ_TOTQUO(R5) : and the total quota
OAB0 3032 BRB 55$ : Continue
00CC C5 59 B0 OAB0 3033 50$: MOVW R9,UCBSW_XQ_QUOTA(R5) : Set current quota
0190 C5 59 B0 OAB0 3034 MOVW R9,UCBSW_XQ_TOTQUO(R5) : and total quota

```

52	2A	C2	OB46	3035	55\$:	SUBL	#SHR_C_LENGTH,R2	:	Backup to beginning of structure	
58	02	91	OB49	3036		CMPB	#NMASC_ACC_LIM,R8	:	Is this for limited use?	
	07	13	OB4C	3037		BEQL	60\$:	Br if YES	
00C4	C5	52	OB4E	3038		MOVL	R2,UCB\$Q_XQ_DEFUSR(R5)	:	Else, save default user address	
	05	11	OB53	3039		BRB	65\$:	Skip linking onto list	
009C	D5	62	OE	OB55	3040	60\$:	INSQUE	(R2),@UCB\$Q_XQ_SHARE+4(R5)	:	Link user into shared user list
	5C	A5	B6	OB5A	3041	65\$:	INCW	UCB\$W_REFC(R5)	:	Increment the ref count on the
				OB5D	3042			:	UCB to be used	
	57	55	D1	OB5D	3043		CMPL	R5,R7	:	Was this the original UCB?
		20	13	OB60	3044		BEQL	70\$:	Br if YES - no more work to do
	5C	A5	B6	OB62	3045		INCW	UCB\$W_REFC(R5)	:	Else, increment REFC (for \$DASSGN)
1C	A3	55	D0	OB65	3046		MOVL	R5,IRP\$Q_UCB(R3)	:	Return new UCB address
	66	55	D0	OB69	3047		MOVL	R5,CCB\$Q_UCB(R6)	:	in CCB also.
		28	BB	OB6C	3048		PUSHR	#*M<R3,R5>	:	Save IRP, real UCB address
	55	57	D0	OB6E	3049		MOVL	R7,R5	:	Copy old UCB address
	5C	A5	B7	OB71	3050		DECW	UCB\$W_REFC(R5)	:	Decrement the reference count
00000000	'GF	16	OB74	3051		JSB	G*IOC\$CREDIT_UCB	:	Restore UCB quota to JIB	
00000000	'GF	16	OB7A	3052		JSB	G*IOC\$DELETE_UCB	:	Delete the old UCB	
		28	BA	OB80	3053		POPR	#*M<R3,R5>	:	Restore IRP, UCB address
	50	01	9A	OB82	3054	70\$:	MOVZBL	S*#SS\$_NORMAL,R0	:	Return success
	0300	8F	BA	OB85	3055	80\$:	POPR	#*M<R8,R9>	:	Restore registers, R4 is PCB address
		0B	BB	OB89	3056		PUSHR	#*M<R0,R1,R3>	:	Save IRP address, status return
00000000	'GF	16	OB8B	3057		JSB	G*SCH\$IOUNLOCK	:	Unlock I/O data base	
	0B	BA	OB91	3058		POPR	#*M<R0,R1,R3>	:	Restore IRP address, status return	
		05	OB93	3059		RSB		:	Return to caller	


```

      OBD3 3116 ; Error on parameter validation
      OBD3 3117 ;
51    F5CC CF42 3C  OBD3 3118 70$:  MOVZWL  BAD_PARAM_TBL-2[R2],R1 ; Return parameter code
      50    14  9A  OBD9 3119 80$:  MOVZBL  S^#SS$_BADPARAM,R0 ; Return bad parameter error
      56    BED0 OBDC 3120 100$: POPL    R6 ; Restore registers
      05    OBDF 3121 RSB
```

```

OBE0 3123      .SBTTL SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
OBE0 3124      :++
OBE0 3125      : SET_MULTIN - SET NEW MULTICAST ADDRESS LIST IN UCB
OBE0 3126      :
OBE0 3127      : Functional description:
OBE0 3128      :
OBE0 3129      : Copy the multicast address list from the old UCB to the new UCB.
OBE0 3130      : This operation is only done for the SHARED DEFAULT user of the PROTOCOL.
OBE0 3131      :
OBE0 3132      : Inputs:
OBE0 3133      :     R5 = UCB address of existing shared user
OBE0 3134      :     R7 = UCB address of new shared user
OBE0 3135      :     R8 = Protocol access mode
OBE0 3136      :
OBE0 3137      : Outputs:
OBE0 3138      :     R0 = Always success
OBE0 3139      :     All other are preserved.
OBE0 3140      :
OBE0 3141      :--
OBE0 3142
OBE0 3143 SET_MULTIN:
58  01  91  OBE0 3144      CMPB   #NMASC_ACC_SHR,R8      ; Set new multicast list
      15  12  OBE3 3145      BNEQ   50$                ; Is this the shared default user?
      3E  BB  OBE5 3146      PUSHR  #*M<R1,R2,R3,R4,R5> ; Br if not
0048 8F  28  OBE7 3147      MOVCS  #6*MAX_C_MLT,UCBSG_XQ_MULTI(R7),- ; Save registers
      C7  OBEB
00EC  C5  OBEF
      3E  BA  OBF1 3148      POPR   UCBSG_XQ_MULTI(R5)      ; Copy multicast list
00E9  C7  90  OBF3 3149      MOVB  UCBSB_XQ_MULTI(R7),- ; Restore registers
00E9  C5  OBF7 3150      MOVB  UCBSB_XQ_MULTI(R7),- ; Copy the number of valid addresses
50  01  9A  OBF9 3151      MOVZBL UCBSB_XQ_MULTI(R5)
      05  OBFD 3152 50$:   MOVZBL #SS$_NORMAL,R0      ; Return success
      RSB                                ; Return to caller

```



```

OBFE 3155      .SBTTL SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
OBFE 3156      :++
OBFE 3157      : SENSEMODE_FDT - SENSEMODE I/O FDT PROCESSING
OBFE 3158      :
OBFE 3159      : Functional description:
OBFE 3160      :
OBFE 3161      : Process read status and read counters requests.
OBFE 3162      :
OBFE 3163      : The QIO parameters for SENSEMODE are:
OBFE 3164      :
OBFE 3165      :     P1 = Optional address of quadword buffer
OBFE 3166      :     P2 = Optional address of buffer descriptor for extended characteristics
OBFE 3167      :
OBFE 3168      : The SUBFUNCTION modifiers are as follows:
OBFE 3169      :
OBFE 3170      : 1) READ PARAMETERS -- NO MODIFIER.
OBFE 3171      :    This function reads the QNA parameters and returns them to the user.
OBFE 3172      :
OBFE 3173      : 2) READ COUNTERS -- !OSM_RD_COUNT SET.
OBFE 3174      :    This function reads the QNA counters and returns them to the user.
OBFE 3175      :
OBFE 3176      : CLEAR COUNTERS -- IOSM_CLR_COUNT SET.
OBFE 3177      :    This modifier must be used with the read counters modifier to clear
OBFE 3178      :    the counters as they are read.
OBFE 3179      :
OBFE 3180      :
OBFE 3181      : Inputs:
OBFE 3182      :
OBFE 3183      :     R3 = IRP address
OBFE 3184      :     R4 = PCB address
OBFE 3185      :     R5 = UCB address
OBFE 3186      :     R6 = CCB address
OBFE 3187      :     R7 = Function code
OBFE 3188      :     AP = Address of first function-dependent QIO parameter
OBFE 3189      :
OBFE 3190      :     IPL = ASTDEL
OBFE 3191      :
OBFE 3192      : Outputs:
OBFE 3193      :
OBFE 3194      :     R0 = Status return of SENSEMODE request
OBFE 3195      :
OBFE 3196      :     R1,R2,R6,R7 are destroyed.
OBFE 3197      :
OBFE 3198      :--
OBFE 3199      :
OBFE 3200      SENSEMODE_FDT::
OBFE 3201      MOVZWL #SS$ DEVOFFLINE,R0      ; SENSE MODE I/O FDT processing
OBFE 3202      BBC      #UCB$V ONLINE,-      ; Assume unit if offline
OBFE 3203      UCBSW_STS(R5),15$      ; Br if unit not online
OBFE 3204      MOVW   IRPSW_FUNC(R3),R7      ;
OBFE 3205      BBS    #IOSV_CTRL,R7,10$      ; Get entire function code
OBFE 3206      :                               ; Br if line request
OBFE 3207      :
OBFE 3208      : Check if read circuit counters
OBFE 3209      BBC    #IOSV_RD_COUNT,R7,5$      ; Br if not read circuit counters
OBFE 3210      BRW   READ_CIRC_CTR      ; Else, get the circuit counters
OBFE 3211      MOVZBL S^#SS$_NORMAL,R0      ; Return success
50 0084 8F 3C OBFE 3201
    04 E1 OC03 3202
    23 64 A5 OC05 3203
57 20 A3 B0 OC08 3204
14 57 09 E0 OC0C 3205
    OC10 3206
    OC10 3207
    OC10 3208
03 57 08 E1 OC10 3209
    00D2 31 OC14 3210
    50 01 9A OC17 3211

```

:RNG0001
:RNG0001
-3

:RNG0001
-1

```

51 44 A5 DO OC1A 3212 MOVL UCBSL_DEVDEPEND(R5),R1 ; Get device dependent information
00000000'GF 17 OC1E 3213 JMP G^EXE$FINISHIC ; Complete the I/O request
06 57 08 E1 OC24 3214 10$: BBC #IOSV_RD_COUNT,R7,20$ ; Br if not read counters
OC28 3215 :
OC28 3216 : Read counters - modifier RD_COUNT
OC28 3217 :
OC28 3218 :
009F 31 OC28 3219 BRW READ_LINE_CTR ; Get the line counters
OC2B 3220 :
712 31 OC2B 3221 15$: BRW ABORTIO ; Abort the I/O request
OC2E 3222 :
OC2E 3223 : Read parameters - no modifier
OC2E 3224 :
OC2E 3225 20$: CLRW IRPSW_XQ_P2SIZ(R3) ; No return data
54 44 A3 B4 OC2E 3225 MOVL UCBSL_XQ_CDB(R5),R4 ; Get CDB address
00C8 C5 DO OC31 .1 ; Size of P1 buffer if present
51 08 3C OC36 .2 ; Check P1 and P2 buffers
017E 30 OC39 3229 BSBW CHECK_BUFS ; Assume success
50 01 9A OC3C 3230 MOVZBL S^#SS$ NORMAL,R0 ; Save user P2 buffer length
3A A3 50 B0 OC3F 3231 MOVW R0,IRPSW_XQ_STATUS(R3) ; Br if no P2 buffer present
38 A3 51 B0 OC43 3232 MOVW R1,IRPSW_XQ_USERSIZ(R3) ; Save user P2 buffer address
47 13 OC47 3233 BEQL 40$ ; Return the P2 parameters
OC49 3234 ; Set size of return data
40 A3 52 DO OC49 3235 MOVL R2,IRPSL_XQ_P2BUF(R3) ; Is CDB present?
1B21 30 OC4D 3236 BSBW RETURN_P2 ; Br if no - okay to return now
44 A3 50 B0 OC50 3237 MOVW R0,IRPSW_XQ_P2SIZ(R3) ; Check if default physical
54 D5 OC54 3238 TSTL R4 ; address can fit
2C 13 OC56 3239 BEQL 30$ ; Is buffer big enough for
51 50 0A C1 OC58 .1 ADDL3 #10,R0,R1 ; default physical address?
OC5C 3241 ; Br if no
38 A3 51 B1 OC5C 3242 CMPW R1,IRPSW_XQ_USERSIZ(R3) ; Add DPA to return buffer size
OC60 3243 ; Get buffer address
OC60 3244 BGTRU 25$ ; past end of return data
44 A3 0A 1A OC60 3244 ADDW #10,IRPSW_XQ_P2SIZ(R3) ; Store parameter code + size
40 A3 50 C1 OC66 3246 ADDL3 R0,IRPSL_XQ_P2BUF(R3),R2 ; and indicate this is a string
OC6A 3247 ; Store Default Physical Address
00061498 8F DO OC6B 3248 MOVL #<6@16>+NMASC_PCLI_HWA!- ; All is okay
82 0250 C4 DO OC72 3249 PRM_TYP M_STRING,(R2)+ ; Return partial success
82 0254 C4 B0 OC77 3251 MOVW CDB_G_HWA+4(R4),(R2)+ ; Get size of user return data
06 11 OC7C 3252 BRB 30$ ; Shift size of buffer return
0601 8F B0 OC7E 3253 25$: MOVW #SS$_BUFFEROVF,IRPSW_XQ_STATUS(R3) ; Get status
50 44 A3 3C OC84 3254 30$: MOVZWL IRPSW_XQ_P2SIZ(R3),R0 ; Is there a CDB?
50 50 10 78 OC88 3255 ASHL #16,R0,R0 ; Br if no CDB yet!
50 3A A3 B0 OC8C 3256 MOVW IRPSW_XQ_STATUS(R3),R0 ; Retrieve P1 buffer address
OC90 3257 ; Br if none
54 D5 OC90 3258 40$: TSTL R4 ; Else, return characteristics
15 13 OC92 3259 BEQL 50$ ; ...
52 3C A3 DO OC94 3260 MOVL IRPSL_XQ_USERBUF(R3),R2 ; Get device dependent info
OF 13 OC98 3261 BEQL 50$ ; ..from UCB also
62 40 A5 7D OC9A 3262 MOVQ UCBSB_DEVCLASS(R5),(R2)
010C C4 C8 OC9E 3263 BISL CDB_L_DEVDEPEND(R4),4(R2)
51 010C C4 DO OCA4 3264 MOVL CDB_L_DEVDEPEND(R4),R1
51 44 A5 C8 OCA9 3265 50$: BISL UCBSL_DEVDEPEND(R5),R1

```

```
00000000'GF 17 0CAD 3266          JMP      G^EXESFINISHIO          ; Complete the I/O request
                OCB3 3267
                OCB3 3268      ;
                OCB3 3269      ; Queue I/O request to driver
                OCB3 3270      ;
                OCB3 3271      ;
                OCB3 3272      ;
00000000'GF 16 0CB7 3273          SETIPL  UCBSB FIPL(R5)          ; Queue packet to driver
00000000'GF 17 0CBD 3274          JSB     G^IOCSINITIATE          ; Raise IPL to fork IPL
                JMP      G^EXESQIORETURN          ; Intiate the I/O request
                ; 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      :--

```

0700 8F BA
F676 31

```

OCC3 3307      .ENABL  LSB
OCC3 3308 ABORT_IRP:
OCC3 3309      POPR    #*M<R8,R9,R10>      ; Restore registers
OCC7 3310      BRW    ABORTIO             ; Abort the I/O request
OCCA 3311

```

58

0700 8F BB
F437 CF 9E

```

OCCA 3312 READ_LINE_CTR:                ; Read the line counters
OCCA 3313      POSHR  #*M<R8,R9,R10>    ; Save registers
OCC3 3314      MOVAB  LINE_CTR,R8       ; Get address of counter format table
OCC3 3315      MOVZWL #LINE_CTR_SIZE,R9 ; Get number of entries in table
OCC3 3316      MOVZWL #LINE_CTR_BUFSIZ,R6 ; Get size of system P2 buffer
OCC3 3317      MOVZWL #SS$ INSMEM,R0     ; Assume no CDB
OCC3 3318      MOVL   UCB$C_XQ CDB(R5),R10 ; Get CDB address
OCC3 3319      BEQL   ABORT_IRP         ; Br if none, abort the I/O
OCC3 3320      BRB    10$              ; Else, Continue in common code
OCC3 3321
OCC3 3322
OCC3 3323 READ_CIRC_CTR:                ; Read the circuit counters
OCC3 3324      POSHR  #*M<R8,R9,R10>    ; Save registers
OCC3 3325      MOVAB  CIRC_CTR,R8       ; Get address of counter format table
OCC3 3326      MOVZWL #CIRC_CTR_SIZE,R9 ; Get number of entries in table
OCC3 3327      MOVZWL #CIRC_CTR_BUFSIZ,R6 ; Get size of system P2 buffer
OCC3 3328      MOVL   R5,R10           ; Use UCB for counters
OCC3 3329
OCC3 3330 10$:      BSBW  CHECK_P2       ; Check the P2 buffer
OCC3 3331      MOVZBL S*#SS$ BADPARAM,R0 ; Assume zero length buffer
OCC3 3332      MOVW  R1,IRP$W_XQ_USERSIZ(R3) ; Save size of user P2 buffer
OCC3 3333      BEQL  ABORT_IRP         ; Br if no buffer

```

:RNG0001
:RNG0001
-3

56

006A 8F 3C
0124 8F 3C

5A

00C8 C5 D0
DC 13 OCE5
12 11 OCE7

58

0700 8F BB
F468 CF 9E

56

06 3C OCF2
20 3C OCF5

5A

55 D0 OCF8
OCFB 3329

38

00BE 30 OCFB
50 14 9A OCFE
A3 51 B0 OD01
BC 13 OD05

```

51 56 D0 OD07 3334      MOVL   R6,R1           ; Get size of system P2 buffer
      00E9 30 OD0A 3335      BSBW   ALLOC_P2BUF      ; Allocate the buffer
      B3 50 E9 OD0D 3336      BLBC   RC,ABORT IRP     ; Br if error
51 2C A3 D0 OD10 3337      MOVL   IRP$&L SVAPTE(R3),R1 ; Get system P2 buffer address
04 A1 52 D0 OD14 3338      MOVL   R2,P2B_L_BUFFER(R1) ; Save user P2 buffer address
      52 61 D0 OD18 3339      MOVL   P2B_L_POINTER(R1),R2 ; Get address of data portion of buffer
      OD1B 3340      ;
      OD1B 3341      ; Get the counters kept by the driver
      OD1B 3342      ;
      50 88 3C OD1B 3343 20$: MOVZWL (R8)+,R0           ; Get counter code
      82 50 B0 OD1E 3344      MOVW   R0,(R2)+          ; Return counter type code
      51 88 3C OD21 3345      MOVZWL (R8)+,R1           ; Get offset word
      51 5A C0 OD24 3346      ADDL   R10,R1           ; Point to counter in UCB
50 03 0C EF OD27 3347      EXTZV #NMA$V_CNT_MAP,#3,R0,R0 ; Get width + bit map indicator
      OD2B 3348      CASE   R0,TYPE=B,LIMIT=#2,<- ; Dispatch on width and bit map
      OD2C 3349      30$,- ; 8 bit counter
      OD2C 3350      30$,- ; 8 bit counter + bit map
      OD2C 3351      40$,- ; 16 bit counter
      OD2C 3352      35$,- ; 16 bit counter + bit map
      OD2C 3353      35$> ; 32 bit counter
      OD3A 3354      ;
      OD3A 3355 30$:  BUG_CHECK NOBUFPCKT,FATAL
      OD3E 3356      ;
      OD3E 3357      ;
      OD3E 3358      ; 32 BIT counter/ 16 BIT counter + bitmap
      OD3E 3359      ;
      03 82 81 B0 OD3E 3360 35$: MOVW   (R1)+,(R2)+          ; Store counter in buffer
      57 0A E1 OD41 3361      BBC    #IO$V_CLR_COUNT,R7,40$ ; Br if not clear counter operation
      FE A1 B4 OD45 3362      CLRW   -2(R1)           ; Else, clear the counter as well
      OD48 3363      ;
      OD48 3364      ; 16 BIT counter
      OD48 3365      ;
      02 82 61 B0 OD48 3366 40$: MOVW   (R1),(R2)+          ; Store counter in buffer
      57 0A E1 OD4B 3367      BBC    #IO$V_CLR_COUNT,R7,50$ ; Br if not clear counter operation
      61 B4 OD4F 3368      CLRW   (R1)           ; Else, clear the counter as well
      C7 59 F5 OD51 3369 50$: SOBGTR R9,20$           ; Loop if more
      51 01 9A OD54 3370      MOVZBL S^#SS$ NORMAL,R1 ; Assume success
      38 A3 B1 OD57 3371      CMPW   IRP$&XQ_USERSIZ(R3),- ; Is user's buffer big enough?
      32 A3 B1 OD5A 3372      CMPW   IRP$&BCNT(R3) ;
      38 A3 IE OD5C 3373      BGEQU  60$           ; Br if yes
      32 A3 B0 OD5E 3374      MOVW   IRP$&XQ_USERSIZ(R3),- ; Else, set size to minimum
      38 A3 B0 OD61 3375      MOVW   IRP$&BCNT(R3) ; Of both
      51 0601 8F B0 OD63 3376 60$: MOVW   #SS$ BUFFEROVF,R1 ; Set partial success
      50 30 A3 D0 OD68 3377      MOVL   IRP$&BCNT-2(R3),R0 ; Get size of buffer returned in
      OD6C 3378      ; ..high word of R0
      50 51 B0 OD6C 3379      MOVW   R1,R0           ; Get status return
      51 44 A5 D0 OD6F 3380      MOVL   UCB$&L_DEVDEPEND(R5),R1 ; Get device dependent info
      0700 8F BA OD73 3381      POPR   #*M<R8,R9,R10> ; Restore registers
00000000'GF 17 OD77 3382      JMP    G^EXE$FINISHIO ; Complete the I/O request
      OD7D 3383      ;
      OD7D 3384      ;
      OD7D 3385      ;
      .DSABL LSB

```

```

OD7D 3387      .SBTTL GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
OD7D 3388      :++
OD7D 3389      : GET_CHAR_BUF - GET P2 CHARACTERISTICS BUFFER
OD7D 3390      :
OD7D 3391      : Functional description:
OD7D 3392      :
OD7D 3393      : This routine saves the P2 buffer for later use by the driver.
OD7D 3394      : The P2 buffer is saved by allocating the appropriate amount of memory from
OD7D 3395      : non-paged pool. The user's quota is checked before the allocation is made.
OD7D 3396      : And the non-paged pool buffer is charged against the user's quota. The P2
OD7D 3397      : system buffer address is passed in IRP$S_SVApte of the IRP.
OD7D 3398      :
OD7D 3399      :
OD7D 3400      : Inputs:
OD7D 3401      :
OD7D 3402      :     R3 = IRP address
OD7D 3403      :     R4 = PCB address
OD7D 3404      :     R5 = UCB address
OD7D 3405      :
OD7D 3406      : Outputs:
OD7D 3407      :
OD7D 3408      :     R0 = status of buffers
OD7D 3409      :
OD7D 3410      :     R3-R5 are preserved.
OD7D 3411      :
OD7D 3412      :--
OD7D 3413      :
OD7D 3414      GET_CHAR_BUF:                                : Get characteristics buffer
OD7D 3415      :
OD7D 3416      : Check access to P2 buffer and check process's buffer quota
OD7D 3417      :
51  04 AC      DO  OD7D 3418 10$: MOVL    P2(AP),R1                : Get address P2 char buf desc
      33      13  OD81 3419      BEQL    40$                    : Br if no P2 buffer
      53      DD  OD83 3420      PUSHL   R3                        : Save R3
00000000'GF  16  OD85 3421      JSB     G^EXE$PROBER_DSC          : Check access to buffer
      OE 50  E9  OD8B 3422      BLBC   R0,15$                    : Br if error
      51 51  3C  OD8E 3423      MOVZWL  R1,R1                    : Get the length as a word
      52      DD  OD91 3424      PUSHL   R2                        : Save R2
00000000'GF  16  OD93 3425      JSB     G^EXE$BUFQUOPRC          : Check for buffered quota
      52 8ED0 OD99 3426      POPL    R2                        : Restore R2
      53 8ED0 OD9C 3427 15$: POPL    R3                        : Restore R3
      01 50  E8  OD9F 3428      BLBS   R0,30$                    : Branch if quota ok
      05      ODA2 3429 20$: RSB                                : Return
      ODA3 3430      :
      ODA3 3431      :
      ODA3 3432      : Quota OKAY, allocate buffer and copy info.
      ODA3 3433      :
      0050 30  ODA3 3434 30$: BSBW    ALLOC P2BUF                : Allocate buffer
      10 50  E9  ODA6 3435      BLBC   R0,50$                    : Br if error
50  2C A3      DO  ODA9 3436      MOVL    IRP$S_SVApte(R3),R0          : Get P2 buffer address
      38      BB  ODAE 3437      PUSHL   #^M<R3,R4,R5>              : Save sacred registers
      62 51  28  ODAF 3438      MOVCL   R1,(R2),P2B_T_DATA(R0)      : Save P2 char buffer
      0C A0      ODB2 3439      :
      38      BA  ODB4 3439      POPR    #^M<R3,R4,R5>              : Restore registers
      50 01  9A  ODB6 3440 40$: MOVZBL  S^SSS_NORMAL,R0          : Set success
      05      ODB9 3441 50$: RSB                                : Return
  
```

```

ODBA 3443      .SBTTL CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
ODBA 3444      :++
ODBA 3445      : CHECK_BUFS - CHECK P1 AND P2 BUFFERS FOR WRITE ACCESS
ODBA 3446      :
ODBA 3447      : Functional description:
ODBA 3448      :
ODBA 3449      : This routines checks the P1 and P2 buffers for write access if supplied.
ODBA 3450      :
ODBA 3451      : Inputs:
ODBA 3452      :
ODBA 3453      :     R1 = Size of P1 buffer needed for write access
ODBA 3454      :     R3 = IRP address
ODBA 3455      :     R4 = PCB address
ODBA 3456      :     R5 = UCB address
ODBA 3457      :     R7 = Function code
ODBA 3458      :
ODBA 3459      : Outputs:
ODBA 3460      :
ODBA 3461      :     R0 is destroyed.
ODBA 3462      :     R1 = Length of P2 buffer (zero if no P2 buffer)
ODBA 3463      :     R2 = Address of P2 buffer in user's process space
ODBA 3464      :
ODBA 3465      :
ODBA 3466      :     No RETURN on NO ACCESS
ODBA 3467      :
ODBA 3468      : Implicit Outputs:
ODBA 3469      :
ODBA 3470      :     IRPSV_FUNC bit set in IRPSW_STS by EXES$READCHK subroutine.
ODBA 3471      :
ODBA 3472      :--
ODBA 3473      :
ODBA 3474      CHECK_BUFS:
27 10 ODBA 3475      BSBB      CHECK_P1      ; Check P1 buffer
ODBC 3476      CHECK_P2:
52 04 AC 00 ODBC 3477      CLRL      R1      ; Assume no P2 buffer desc
18 13 ODBE 3478      MOVL      P2(AP),R2 ; Get address of P2 desc
ODC2 3479      BEQL      10$      ; Br if no P2
ODC4 3480      IFNORD   #8,(R2),ACCESS ; Br if no access
51 62 3C ODCA 3481      MOVZWL   (R2),R1 ; Get length of buffer
OD 13 ODCD 3482      BEQL      10$      ; Br if zero
50 04 A2 00 ODCF 3483      MOVL      DSC$A_POINTER(R2),R0 ; Get buffer address
00000000'GF 16 ODD3 3484      JSB      G^EXES$READCHK ; Check write access to buffer
ODD9 3485      ; (no return no access)
ODD9 3486      ; Also sets IRPSV_FUNC in IRP
52 50 00 ODD9 3487      MOVL      R0,R2 ; Copy buffer address
05 ODDC 3488      10$:      RSB ; Return to caller
ODDD 3489      ;
50 0C 9A ODDD 3490      ACCESS: MOVZBL   S^#SS$ ACCVIO,R0 ; Return access violation
F55D 31 ODE0 3491      BRW      ABORTIO ; Abort the I/O request

```

```

ODE3 3493      .SBTTL CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
ODE3 3494      :++
ODE3 3495      : CHECK_P1 - CHECK P1 BUFFER ADDRESS FOR WRITE ACCESS
ODE3 3496      :
ODE3 3497      : Functional description:
ODE3 3498      :
ODE3 3499      : This routine checks the P1 buffer and if okay, the buffer address
ODE3 3500      : is saved in IRP$L_MEDIA of the IRP.
ODE3 3501      :
ODE3 3502      : Inputs:
ODE3 3503      :
ODE3 3504      :     R1 = Size of buffer for write access
ODE3 3505      :     R3 = IRP address
ODE3 3506      :     R4 = PCB address
ODE3 3507      :     R5 = UCB address
ODE3 3508      :     R7 = Function code
ODE3 3509      :
ODE3 3510      : Outputs:
ODE3 3511      :
ODE3 3512      :     R0 is destroyed.
ODE3 3513      :
ODE3 3514      :     No RETURN on NO ACCESS.
ODE3 3515      :
ODE3 3516      : Implicit Outputs:
ODE3 3517      :
ODE3 3518      :     IRP$L_MEDIA(R3) = User P1 buffer address.
ODE3 3519      :     IRP$V_FUNC bit set in IRP$W_STS by EXE$READCHK subroutine.
ODE3 3520      :
ODE3 3521      :--
ODE3 3522      :
ODE3 3523      CHECK_P1:
   38 A3      D4 ODE3 3524      CLR    IRP$L_MEDIA(R3)           : Assume no P1 buffer
   50 6C      D0 ODE6 3525      MOVL   P1(AP),R0             : Get address of user buffer
   00000000 GF 13 ODE9 3526      BEQL   10$                  : Br if none
   3C A3      50 ODEB 3527      JSB    G^EXE$READCHK         : Check access to buffer
   05 ODF1    3528      ODF1 3528      : (No return - no access)
   05 ODF1    3529      10$: MOVL   R0,IRP$L_XQ_USERBUF(R3)    : Save P1 buffer address in IRP
   05 ODF5    3530      RSB                    : Return to caller

```



```

ODF6 3532 .SBTTL ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
ODF6 3533 :++
ODF6 3534 : ALLOC_P2BUF - ALLOCATE A P2 BUFFER AND CHARGE USER'S QUOTA
ODF6 3535 :
ODF6 3536 : Functional description:
ODF6 3537 :
ODF6 3538 : This routine allocates a system buffer and returns the address in the IRP at
ODF6 3539 : IRP$S_SVAPTE. The size of the allocation, including buffer header must
ODF6 3540 : be at least 24 bytes in length.
ODF6 3541 :
ODF6 3542 : Inputs:
ODF6 3543 :
ODF6 3544 : R1 = Size of allocation desired
ODF6 3545 : R3 = IRP address
ODF6 3546 :
ODF6 3547 : Outputs:
ODF6 3548 :
ODF6 3549 : R0 = status of request
ODF6 3550 :
ODF6 3551 : R1-R5 are preserved.
ODF6 3552 :
ODF6 3553 : Implicit Outputs:
ODF6 3554 :
ODF6 3555 : IRP$S_SVAPTE(R3) = address of system buffer
ODF6 3556 : IRP$W_BOFF(R3) = byte count charged to user's process
ODF6 3557 : IRP$W_BCNT(R3) = original byte count requested
ODF6 3558 :
ODF6 3559 : All parts of the P2 buffer header are initialized, except for the
ODF6 3560 : user's P2 buffer address.
ODF6 3561 :
ODF6 3562 :--

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
-4

```

ODF6 3564 ALLOC_P2BUF:
ODF6 3565 TSTL R1 ; Allocate a non-paged buffer
ODF6 3566 BEQL 30$ ; Zero length buffer?
ODF6 3567 PUSHF #^M<R1,R2,R3> ; Br if yes
ODF6 3568 MOVW R1,IRP$W_BCNT(R3) ; Save registers
ODF6 3569 CMPL R1,#24-P2B_C_LENGTH ; Save original byte count
ODF6 3570 BGTRU 5$ ; Is buffer big enough?
ODF6 3571 MOVL #24-P2B_C_LENGTH,R1 ; Br if yes
ODF6 3572 ADDL2 #P2B_C_LENGTH,R1 ; Else, set size to minimum
ODF6 3573 JSB G^EXE$BUFQUOPRC ; Add in size of header
ODF6 3574 BLBC R0,10$ ; Check for buffered quota
ODF6 3575 ; Branch if quota bad
ODF6 3576 : Quota OKAY, allocate buffer and copy info.
ODF6 3577 :
ODF6 3578 PUSHL R1 ; Save size to charge user
ODF6 3579 JSB G^EXE$ALLOCBUF ; Go allocate a buffer
ODF6 3580 BLBS R0,20$ ; Br if success
ODF6 3581 ADDL #4,SP ; Pop saved size
ODF6 3582 POPR #^M<R1,R2,R3> ; Restore registers
ODF6 3583 RSB ; Return with error code in R0
ODF6 3584 :
ODF6 3585 : System buffer allocated decrement user's quota
ODF6 3586 :
ODF6 3587 20$: POPL R3 ; Restore user quota charge
ODF6 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\$JIB(R4),R2	:	Get JIB address
20	A2	53	C2	0E38	3592	SUBL	R3,JIB\$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\$SVAPTE(R3)	:	Save P2 buffer address in IRP
30	A3	08	B0	0E42	3595	MOVW	P2B_W_SIZE(R0),IRP\$W_BOFF(R3)	:	Return buffer size in IRP
	50	01	9A	0E47	3596	MOVZBL	S#SS\$NORMAL,R0	:	Set success
			05	0E4A	3597	RSB		:	Return to caller

```

OE4B 3599      .SBTTL  STARTIO - START I/O OPERATION
OE4B 3600      :++
OE4B 3601      : STARTIO - START I/O OPERATION
OE4B 3602      :
OE4B 3603      : Functional description:
OE4B 3604      :
OE4B 3605      : This routine is called when an IRP is ready to be processed by the driver.
OE4B 3606      : The request is dispatched to the appropriate routine base on the internal
OE4B 3607      : function code in the IRP.
OE4B 3608      :
OE4B 3609      : Inputs:
OE4B 3610      :
OE4B 3611      :     R3 = IRP address
OE4B 3612      :     R5 = UCB address
OE4B 3613      :
OE4B 3614      :     IPL = FIPL
OE4B 3615      :
OE4B 3616      : Outputs:
OE4B 3617      :
OE4B 3618      :     R0-R2,R4 are destroyed.
OE4B 3619      :--
OE4B 3620      :
OE4B 3621      STARTIO::
OE4B 3622      : Process an I/O packet
OE4B 3623      : Get CDB address
OE4B 3624      : Get the internal function code
OE50 3624      MOVBL  UCBSL_XQ_CDB(R5),R4
OE54 3625      10$:  $DISPATCH  R1,TYPE=B,-
OE54 3626      <-      ;function      action
OE54 3627      :
OE54 3628      <XQ_FC_V_INIT  STARTUP>,-      ; Startup request
OE54 3629      <XQ_FC_V_STOP  SHUT>,-      ; Shutdown request
OE54 3630      <XQ_FC_V_CHMODE CHMODE>,-      ; Set new multicast list
OE54 3631      >
OE66 3632      :
OE66 3633      : Other request type
OE66 3634      :
OE66 3635      :     BUG_CHECK NOBUFCKT,FATAL      ; Fatal error
OE6A 3636      :
OE6A 3637      :
OE6A 3638      : Startup unit's protocol
OE6A 3639      :
OE6A 3640      STARTUP:
OE6A 3641      : Startup unit's protocol
OE6A 3642      : Start protocol
OE6C 3642      BSBB  START
OE6F 3643      BLBC  RO,10$
OE70 3644      RSB
OE70 3645      10$:  PUSH  RO      ; Save error return
OE72 3646      BSBB  STOP      ; Shutdown unit
OE74 3647      POPL  RO      ; Restore error return
OE77 3648      BRW   IO_DONE   ; Complete the I/O request
OE7A 3649      :
OE7A 3650      :
OE7A 3651      : Shutdown UNIT's protocol
OE7A 3652      :
OE7A 3653      SHUT:
OE7A 3654      : Shutdown protocol
OE7A 3654      Cmpl  R5,CDB_L_PRMUSER(R4) ; Are we the PROMISCUOUS user?
OE7F 3655      BNEQ  10$      ; Br if not
OE81 3656      CLRL  CDB_L_PRMUSER(R4) ; Else, clear the PROMISCUOUS user

```

:RNG0001
-2

54 00C8 C5 D0
51 21 A3 9A

01 49 10
50 E9
05

50 DD
2A 10
50 8ED0
OCD6 31

0210 C4 55 D1
14 12
0210 C4 D4

:RNG0001
-2

```

01 90 0E85 3657      MOVB      #NMA$C_STATE_OFF,-      ; Don't forget about the CDB
0247 C4 0E87 3658      CDB_B_PRM(R4)      ; parameter
04D7 30 0E8A 3659      BSBW      SETUP_MODE      ; Get setup buffer
0D 50 E9 0E8D 3660      BLBC      RO,90$      ; Br if error
03 90 0E90 3661      MOVB      #XQ_FC_V_STOP,-      ; Set function request
20 A2 0E92 3662      CXBSB_XQ_FUNC(R2)  ;
05 05 0E94 3663      RSB      ; Return to complete function
07 10 0E95 3664      0E95 3664      ;
50 01 9A 0E97 3666      10$: BSBW      STOP      ; Shutdown unit
0CB3 31 0E9A 3667      MOVZBL   S^#SS$ NORMAL,RO  ; Return success
0E9D 3668      BRW      IO_DONE      ; Complete I/O request
05 05 0E9D 3669      90$: RSB      ; Return to caller
0E9E 3670      ;
0E9E 3671      ; STOP the unit
0E9E 3672      ;
0E9E 3673      STOP:      ; Stop the protocol
00C8 C5 D0 0E9E 3674      1      ; Get CDB address
1178 31 0EA3 3676      MOVL     UCBSL_XQ_CDB(R5),R4 ; Shutdown the unit
0EA6 3677      BRW      SHUTDOWN_PROTYP
0EA6 3678      ; CHMODE - set new multicast list
0EA6 3679      ;
0EA6 3680      CHMODE:
04BB 30 0EA6 3681      BSBW      SETUP_MODE      ; Get XMIT setup buffer
06 50 E9 0EA9 3682      BLBC      RO,10$      ; Exit if error
06 90 0EAC 3683      MOVB      #XQ_FC_V_CHMODE,-  ; Set function request
20 A2 0EAE 3684      CXBSB_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

```


:RNG0001
:RNG0001
:RNG0001
:RNG0001
-5

```

      OC A3  DO  OEFB 3745      MOVL  IRP$$_PID(R3),UCB$_XQ_PID(R5) ; Save starter's PID
      00B8 C5      OEFB
      OEFE 3746      :
      OEFE 3747      : Check for CDB
      OEFE 3748      :
      52 00C8 C5  DO  OEFE .1 8$: MOVL  UCB$_XQ_CDB(R5),R2      ; Get CDB address
      020B C2  96  OF03 .2      INCB  CDB_B_UNT(CNT(R2))      ; One more unit on this controller
      :RNG0001 03 0246 C2  EB  OF07 .3      ASSUME CDB_STS_V_INITED EQ 0
      :RNG0001 0093 31  OF07 .4      BLBS  CDB_B_STSTR(2),10$      ; Br if already initied
      -5      OF0C 3754      BRW   20$      ; Else, init CDB
      OF0F 3755
      50 54 52  DO  OF0F 3756 10$: MOVL  R2,R4      ; Copy CDB address
      00E1 C5  9E  OF12 3757      MOVAB UCB$_XQ_CDBPRM(R5),R0      ; Get UCB parameter address
      51 0249 C4  9E  OF17 3758      MOVAB CDB_B_SETPRM(R4),R1      ; Get CDB parameter address
      52 01 9A  OF1C 3759      MOVZBL #UCB$_XQ_CDBPRM,R2      ; Set size of parameter list
      OF1F 3760
      OF1F 3761      :
      OF1F 3762      : Check order of UCB parameters
      OF1F 3763      :
      OF1F 3764      ASSUME  UCB$_XQ_CON EQ UCB$_XQ_CDBPRM
      OF1F 3765      :
      OF1F 3766      : Check order of CDB parameters
      OF1F 3767      :
      OF1F 3768      ASSUME  CDB_B_CON EQ CDB_B_SETPRM
      OF1F 3769
      OF1F 3770      ASSUME  NMASC_STATE_ON NE -1
      OF1F 3771      ASSUME  NMASC_STATE_OFF NE -1
      61 60 91  OF1F 3772 13$: CMPB  (R0),(R1)      ; Match?
      73 12  OF22 3773      BNEQ  12$      ; Br if no
      80 81 90  OF24 3774      MOVB  (R1)+,(R0)+      ; Store CDB value in UCB
      F5 52 F5  OF27 3775      SOBGTR R2,13$      ; Loop if more to check
      OF2A 3776      :
      OF2A 3777      : NOW, check if user has given a hardware physical address.
      OF2A 3778      :
      OF2A 3779      : R0 = Address of parameters in UCB
      OF2A 3780      : R1 = Address of parameters in CDB
      OF2A 3781      :
      OF2A 3782      ASSUME  CDB_G_PHA EQ CDB_B_CON+1
      OF2A 3783      ASSUME  UCB$_XQ_PHA EQ UCB$_XQ_CON+1
      FFFFFFFF 8F  D1  OF2A 3784      CMLP  #-1,(R0)      ; Is user physical address defined?
      60  OF30
      08 12  OF31 3785      BNEQ  15$      ; Br if yes
      FFFF 8F  B1  OF33 3786      CMPW  #-1,4(R0)      ; Is user physical address defined?
      04 A0  OF37
      10 13  OF39 3787      BEQL  16$      ; Br if not
      0B04 8F  B0  OF3B 3788 15$: MOVW  #NMASC_PCLI_PHA,IRP$_XQ_CODE(R3) ; Assume bad physical address
      40 A3  OF3F
      81 80  D1  OF41 3789      CMLP  (R0)+,(R1)+      ; Physical address match??
      58 12  OF44 3790      BNEQ  19$      ; Br if no
      81 80  B1  OF46 3791      CMPW  (R0)+,(R1)+      ; Still match??
      53 12  OF49 3792      BNEQ  19$      ; Br if no
      024A C4  DO  OF4B 3793 16$: MOVL  CDB_G_PHA(R4),-      ; Return hardware set address
      00E2 C5  OF4F 3794      UCBS$_XQ_PHA(R5)      ; just in case we defaulted
      024E C4  B0  OF52 3795      MOVW  CDB_G_PHA+4(R4),-      :
      00E6 C5  OF56 3796      UCBS$_XQ_PHA+4(R5)      :
      OF59 3797      :
      OF59 3798      : Check users buffer size - must not be more than twice the hardware buffer

```

```

OF59 3799 ; size. (Already has been checked against max message size).
OF59 3800 ;
OAF1 8F B0 OF59 3801 MOVW #NMASC_PCLI_BUS,IRPSW_XQ_CODE(R3) ; Assume bad buffer size
  40 A3 OF5D
51 0108 C4 3C OF5F 3802 MOVZWL CDB_W_BSZ(R4),R1 ; Get device buffer size
05 00E0 C5 E8 OF64 3803 BLBS UCBSB_XQ_DCH(R5),17$ ; Br if user can't do data chaining
  OF69 3804 ; && Maybe this is an Internal IRP user
51 0108 C4 A0 OF69 3805 ADDW CDB_W_BSZ(R4),R1 ; Compute twice the normal buffer size
  42 A5 B1 OF6E 3806 17$: CMPW UCBSW_DEVBUSIZ(R5),R1 ; Is buffer size okay?
  2A 1A OF72 3807 BGTRU 19$ ; Br if too large
  40 A3 B4 OF74 3808 CLRW IRPSW_XQ_CODE(R3) ; No bad parameters
  10 AB OF77 3809 BISW #UCBSM_XQ_RUN,- ; Indicate we have entered RUN mode
  68 A5 OF79 3810 UCBSW_DEVSTS(R5)
  1589 30 OF7B 3811 BSBW MOVE MULTI ; Copy multicast address list
  OF7E 3812 ASSUME NMASC_STATE_ON EQ 0
  OF7E 3813 ASSUME NMASC_STATE_OFF EQ 1
07 00DE C5 E8 OF7E 3814 BLBS UCBSB_XQ_PRM(R5),173$ ; Br if not promiscuous
  00DE C5 90 OF83 3815 MOVW UCBSB_XQ_PRM(R5),- ; Else, enable promiscuous mode
  0247 C4 OF87 3816 CDB_B_PRM(R4)
  03D7 30 OF8A 3817 173$: BSBW SETUP_MODE ; Allocate setup mode buffer
  06 50 E9 OF8D 3818 BLBC R0,175$ ; Exit if error
  06 90 OF90 3819 MOVW #XQ_FC_V_CHMODE,- ; Set function request
  20 A2 OF92 3820 CXBSB_XQ_FUNC(R2)
  9E 16 OF94 3821 JSB @($P)+ ; Complete request
  OF96 3822 175$: RSB ; Return to caller
  OF97 3823 ;
  OF97 3824 ; Error on parameter validation
  OF97 3825 ;
F208 CF42 B0 OF97 3826 18$: MOVW BAD_PARAM_TBL-2[R2],IRPSW_XQ_CODE(R3) ; Return parameter code
  40 A3 OF9C
  50 14 9A OF9E 3827 19$: MOVZBL S^#SS$_BADPARAM,R0 ; Return bad parameter error
  05 OFA1 3828 RSB ; Return to caller
  OFA2 3829 ;
  OFA2 3830 ; Initialize CDB
  OFA2 3831 ;
51 020A 8F 3C OFA2 3832 20$: MOVZWL #CDB_C_ZERO,R1 ; Get portion of CDB to init with zero
  3E BB OFA7 3833 PUSHR #^M<R1,R2,R3,R4,R5> ; Save registers
00 62 00 2C OFA9 3834 MOVCS #0,(R2),#0,R1,(R2) ; Zero the structure
  62 51 OFAD
  3E BA OFAF 3835 POPR #^M<R1,R2,R3,R4,R5> ; Restore registers
  OFB1 3836 ASSUME CDB_L_FQFL EQ 0
  OFB1 3837 ASSUME CDB_L_FQBL EQ CDB_L_FQFL+4
  54 82 7E OFB1 3838 MOVAQ (R2)+,R4 ; Skip link pointers, copy CDB address
  OFB4 3839 ASSUME CDB_W_SIZE EQ CDB_L_FQBL+4
  82 51 B0 OFB4 3840 MOVW R1,(R2)+ ; Store size of structure
  OFB7 3841 ASSUME CDB_B_TYPE EQ CDB_W_SIZE+2
  OFB7 3842 ASSUME CDB_B_FIPL EQ CDB_B_TYPE+1
82 0833 8F B0 OFB7 3843 MOVW #<IPL$ XQ_FIPL@8>?DYN$C_CDB,(R2)+ ; Set structure type and FIPL
  OFBC 3844 ASSUME CDB_L_FPC EQ CDB_B_FIPL+1
00001663'EF 9E OFBC 3845 MOVAB FORK_PROC,(R2)+ ; Set fork process address
  82 OFC2
  OFC3 3846 ASSUME CDB_L_FR3 EQ CDB_L_FPC+4
  OFC3 3847 ASSUME CDB_L_FR4 EQ CDB_L_FR3+4
  82 7C OFC3 3848 CLRQ (R2)+ ; Clear fork R3 and R4
  OFC5 3849 ASSUME CDB_B_NEXTXMT EQ CDB_L_FR4+4
  OFC5 3850 ASSUME CDB_B_NEXTRCV EQ CDB_B_NEXTXMT+1
  OFC5 3851 ASSUME CDB_B_RCVMAP EQ CDB_B_NEXTRCV+1

```

:RNG0001
-1

```

      82  D4  OFC5 3852 ASSUME CDB_B_XMTMAP EQ CDB_B_RCVMAP+1
      OFC5 3853 CLRRL (R2)+ ; Clear slot in use flags
      OFC7 3854 ASSUME CDB_L_RCVMAP EQ CDB_B_XMTMAP+1
      OFC7 3855 ASSUME CDB_L_XMTMAP EQ CDB_L_RCVMAP+<4*MAX_C_RCV-1>>
51  0A  9A  OFC7 3856 MOVZBL #<MAX_C_XMT-1>+<MAX_C_RCV-1>,R1 ; Set number of mapping vectors
82  01  CE  OFCA 3857 30$: MNEGL #1,(R2)+ ; Indicate no mapping info
      FA 51  F5  OFCD 3858 SOBGTR R1,30$ ; Loop if more
00000098 8F  C0  OFD0 3859 ASSUME CDB_L_RRINGPA EQ CDB_L_XMTMAP+<4*MAX_C_XMT-1>>
      52  OFD0 3860 ADDL #CDB_Q_QUEUES-CDB_L_RRINGPA,R2 ; Skip ring entry pointers
      51  05  9A  OFD7 3861 MOVZBL #CDB_C_QUEUES,R1 ; Set number of queue listheads
82  62  DE  OFDA 3862 40$: MOVAL (R2),(R2)+ ; Set forward link
      FC A2  D0  OFD9 3863 MOVL -4(R2),(R2)+ ; Set backward link
      F6 51  F5  OFE1 3864 SOBGTR R1,40$ ; Loop if more listheads
0246 C4  01  88  OFE4 .1 BISB #CDB_STS_M_INITED,CDB_B_STS(R4) ; Set initial status bits
      OFE9 3866 ;
      OFE9 3867 ; Initialize CDB defaults
      OFE9 3868 ;
00DA C5  B0  OFE9 3869 MOVW UCBSW_XQ_BSZ(R5),- ; Init CDB buffer size
0108 C4  OFED 3870 CDB_W_BSZ(R4)
00D6 C5  B0  OFF0 3871 MOVW UCBSW_XQ_HBQ(R5),CDB_W_QUOTA(R4) ; Set initial quota
010A C4  OFF4
      OFF7 3872
      OFF7 3873 ASSUME INIT_C_AQUOTA LE 255
      OFF7 3874 ASSUME CDB_B_AQUOTA EQ CDB_B_AQUOTA+1
0200 8F  B0  OFF7 3875 MOVW #<INIT_C_AQUOTA>, ; Initialize Maximum QUOTA
02EE C4  OFFB 3876 CDB_B_AQUOTA(R4) ; and zero Additional QUOTA
      OFFE 3877 ASSUME UCBSB_XQ_MLT EQ UCBSB_XQ_PRM+1
      OFFE 3878 ASSUME CDB_B_MLT EQ CDB_B_PRM+1
00DE C5  B0  OFFE 3879 MOVW UCBSB_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 UCBSB_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 #UCBS_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 UCBSW_BCNT(R5)
      1021 3889 PUSHQ R6 ; Save R6,R7
54  56  54  D0  1024 3890 MOVL R4,R6 ; Save CDB address
      24 A5  D0  1027 3891 MOVL UCBSL_CRB(R5),R4 ; Get CRB address
      2C B4  D0  102B .1 ASSUME IDBSL_CSR EQ 0
      0114 C6  102B .2 MOVL @CRBS[INTD+VECSL_IDB(R4)],- ; Save CSR address
57  28 A5  D0  102F .3 CDB_L_CSR(R6)
      04 A7  D0  1031 3892 MOVL UCBSL_DDB(R5),R7 ; Get DDB address
      0110 C6  D0  1035 3893 MOVL DDBSL_UCB(R7),CDB_L_UCB0(R6) ; Set UCB #0 address
      1038
      103B 3894
      103B 3895 CPUDISP <<790,50$>,-
      103B 3896 <<780,50$>,-
      103B 3897 <<750,50$>,-
      103B 3898 <<730,50$>,-
      103B .1 <<UV2,50$>,-
      103B .2 <<UV1,70$>>
      1057 .3

```

:RNG0001
:RNG00C1
:RNG0001

:RNG0001
:RNG0001
:RNG0001

:RNG0001
:RNG0001
:RNG0001
-2

```

1057 .4 50$: ; All but MicroVAX I
1057 .5
1057 .6
1057 3901 ; Allocate map registers for receive buffers and one transmit buffer.
1057 3902
1057 3903 MOVW #511,UCBSW_BOFF(R5) ; Set worst case byte offset
105B
105D 3904 ASSUME VEC$W_MAPREG+2 EQ VEC$B_NUMREG
105D 3905 ASSUME VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
105D 3906 CLRL CRB$L_INTD+VEC$W_MAPREG(R4) ; Clear map register + datapath
56 34 A4 D4 1060 3907 MOVAL CDB_L_RCVMAP(R6),R6 ; Get mapping slot address
57 1C A6 DE 1064 3908 MOVZBL #MAX_C_RCV-1,R7 ; Get number of receive slots
00000000'GF 16 1067 3909 55$: 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
1077 3914 MOVZWL #MAX_PKT_SIZE+18,- ; Set transmit buffer size to max
7E A5 107B 3915 UCBSW_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
66 34 A4 D0 1086 3918 ASSUME CDB_L_XMTMAP EQ CDB_L_RCVMAP+<4*<MAX_C_RCV-1>>
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$L_AUXSTRUC(R4),R6 ; Get CDB address
108E 3924
108E 3925 ASSUME CDB_G_MAPPED EQ CDB_G_RRING
108E 3926 ASSUME CDB_G_XRING EQ CDB_G_RRING+RCV_K_LENGTH
57 0162 C6 9E 108E 3927 MOVAB CDB_G_MAPPED(R6),R7 ; Get starting ring address
FE00 8F AB 1093 3928 BICW3 #^C<VASM_BYTE>,- ; Get PCBB byte offset
7C A5 57 1097 3929 R7,UCBSW_BOFF(R5)
00A8 8F B0 109A 3930 MOVW #CDB_C_MAPPED,UCBSW_BCNT(R5) ; Set Block size
7E A5 109E
00000000'GF 16 10A0 3931 JSB G^IOC$ALOUBAMAP ; Allocate map registers
09 50 E8 10A6 3932 BLBS R0,65$ ; Br if allocated
50 0344 8F 3C 10A9 3933 60$: POPQ R6 ; Restore R6, R7
05 10B1 3934 MOVZWL #SS$_INSFMAPREG,R0 ; Set insufficient map registers
10B2 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$W_MAPREG(R4),- ; Save RING mapping info
011C C6 10B5 3940 CDB_L_RINGMAP(R6)
10B8 3941
10B8 3942 ; Convert the virtual RING address to a UNIBUS mapped address
10B8 3943
51 57 09 EF 10B8 3944 EXTZV S^#VASV_VPN,- ; Get RING page number
57 15 10BA 3945 S^#VASS_VPN,R7,R1
00000000'GF DO 10BD 3946 MOVL G^MMG$GC_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
51 7C A5 3C 10CF .1 MOVZWL UCBSW_BOFF(R5),R1 ; Set BA0-BA8
34 A4 F0 10D3 .2 INSV CRB$L_INTD+VEC$W_MAPREG(R4),- ; Set BA9-BA15
51 0D 09 10D6 .3 #9,#13,R1

```

:RNG0001
:RNG0001
:RNG0001

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-10

```

22 11 10D9 .4 BRB 80$ ; Continue
      10DB .5
      10DB .6 70$: ; MicroVAX I
      10DB .7
      10DB .8
      10DB .9 ; Compute physical address to start of rings for MicroVAX I.
      10DB 3959
      10DB 3960 ASSUME CDB_G_MAPPED EQ CDB_G_RRING
      10DB 3961 ASSUME CDB_G_XRING EQ CDB_G_RRING+RCV_K_LENGTH
57 0162 C6 9E 10DB 3962 MOVAB CDB_G_MAPPED(R6),R7 ; Get starting ring address
57 15 09 EF 10E0 3963 EXTZV #VASS_VPN,#VASS_VPN,R7,R1 ; Get virtual page number
      10E4
00000000 GF DO 10E5 3964 MOVL G^MMG$GL_SPTBASE,R0 ; Get base address of SPTs
      50 10EB
      50 6041 DO 10EC 3965 MOVL (R0)[R1],R0 ; Get the PTE contents
FFFFFE00 8F CB 10F0 3966 BICL3 #^C<VASM_BYTE>,R7,R1 ; Get buffer offset (BA00-BA08)
      51 57 10F6
0D 09 50 FO 10F8 3967 ASSUME PTESS_PFN GE 13
      51 10F8 3968 INSV R0,#9,#13,R1 ; Copy BA09-BA21
      10FC
      10FD 3969
      10FD 3970 80$: ;
      10FD 3971 ; Now calculate the physical/virtual address of each of the ring buffer
      10FC 3972 ; entries, saving the addresses in the PHYSICAL ADDRESS/VIRTUAL
      10FD 3973 ; ADDRESS VECTOR
      10FD 3974
50 0162 C6 9E 10FD 3975 MOVAB CDB_G_RRING(R6),R0 ; Get first ring entry
      57 D4 1102 3976 CLRL R7 ; Use R7 as ring index
44 A647 51 DO 1104 3977 90$: 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 DO 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 DO 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
      1123 3988 CLRL R7
68 A647 51 DO 1125 3989 100$: MOVL R1,CDB_L_XRINGPA(R6)[R7]; Save RING physical address
      51 0C C0 112A 3990 ADDL #XMT_C_LENGTH,R1 ; Skip to next
009C C647 50 DO 112D 3991 MOVL R0,CDB_L_XRINGVA(R6)[R7]; Save RING entry virtual address
      50 0C C0 1133 3992 ADDL #XMT_C_LENGTH,R0 ; Skip to next
      EB 57 04 F2 1136 3993 AOBLSS #MAX_C_XMT,R7,100$ ; Loop if more
68 A647 51 DO 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
      113F 3998 MGVZBL #MAX_C_RCV,R7 ; Set number of entries in ring
51 57 08 9A 113F 3998 MOVAB CDB_G_RRING(R6),R1 ; Get address of RING buffer
      0162 C6 9E 1142 3999 MOVW #RCV_FLG_M_LAST,- ; Init flags
      8000 8F B0 1147 4000 110$: RCV_Q_FLAG(R1)
      61 114B 4001
      08 A1 B4 114C 4002 CLRW RCV_W_STS(R1) ; Zero status
      51 0C C0 114F 4003 ADDL #RCV_C_LENGTH,R1 ; Skip to next entry
      F2 57 F5 1152 4004 SOBGTR R7,1T0$ ; 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_Q_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
      1165 4016 RCV_W_ADDRHI(R1) ;
C000 8F 1168 ;
      02 A1 116A 4017 ;
      116A 4018 ; Initialize transmit ring buffer entries
      116A 4019 ;
51 57 04 9A 116A 4020 MOVZBL #MAX_C_XMT,R7 ; Set number of XMIT entries
01CE C6 9E 116D 4021 MOVAB CDB_G_XRING(R6),R1 ; Get address of RING buffer
8000 8F B0 1172 4022 120$: MOVW #XMT_FLG_M_LAST,- ; Init flags
      61 1176 4023 XMT_Q_FLAG(R1) ;
      08 A1 B4 1177 4024 CLRW XMT_W_STS(R1) ; Zero status
51 0C C0 117A 4025 ADDL #XMT_C_LENGTH,R1 ; Skip to next entry
      F2 57 F5 117D 4026 SOBGTR R7,120$ ; 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_Q_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
      1190 4038 XMT_W_ADDRHI(R1) ;
C000 8F 1193 ;
      02 A1 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
50 0124 8F 3C 11B8 4051 POPQ R6 ; Restore R6, R7
      05 11B8 4052 MOVZWL #SS$_INSMEM,R0 ; Else, return error
      11C0 4053 RSB ;
      11C1 4054 ;
      11C1 4055 123$: ;

```

```

;RNG0001      11C1 4056      ; Compute physical/virtual address of buffers in buffer area
               11C1      ;
               52  0C A2  9E 11C1 4057      MOVAB 12(R2),R2      ; Get start of buffer area
               52  15 09  EF 11C5 4058      EXTZV #VASV_VPN,#VASS_VPN,R2,R0 ; Get virtual page number
               00000000'GF  DO 11CA 4059      MOVL  G^MMG$GL_SPTBASE,R1      ; Get base address of SPTs
               50  6140  DO 11D0      ;
               FFFFEE00 8F  CB 11D1 4060      MOVL  (R1)[R0],R0      ; Get the PTE contents
               51  52      11D5 4061      BICL3 #^C<VASM_BYTE>,R2,R1 ; Get buffer offset (BA00-BA08)
               0D  09  50  FO 11DD 4062      ASSUME PTE$S_PFN GE 13
               51      11DD 4063      INSV  R0,#9,#13,R1      ; Copy BA09-BA21
               50  D4 11E2 4064      CLRL  R0      ; Use R0 as receive buffer index
;RNG0001      00AC C640 51  DO 11E4      .1 125$: MOVL  R1,CDB_L_RCV_PA(R6)[R0] ; Save receive physical address
;RNG0001      00C4 C640 52  DO 11EA      .2      MOVL  R2,CDB_L_RCV_VA(R6)[R0] ; Save receive virtual address
;RNG0001      000005EA 8F  CO 11F0      .3      ADDL  #MAX_BUF$SIZ_OV1,R1      ; Skip to next buffer
;RNG0001      000005EA 8F  CO 11F7      .4      ADDL  #MAX_BUF$SIZ_UV1,R2      ;
;RNG0001      E2  50  05  F2 11FE      .5      AOBLS #MAX_C_RCVUV1,R0,125$ ; Loop if more
;RNG0001      00C0 C640 51  DO 1202      .6      CLRL  R0      ; Use R0 as transmit buffer index
;RNG0001      00D8 C640 52  DO 1204      .7 130$: MOVL  R1,CDB_L_XMT_PA(R6)[R0] ; Save transmit physical address
;RNG0001      000005EA 8F  CO 120A      .8      MOVL  R2,CDB_L_XMT_VA(R6)[R0] ; Save transmit virtual address
;RNG0001      000005EA 8F  CO 1210      .9      ADDL  #MAX_BUF$SIZ_OV1,R1      ; Skip to next buffer
;RNG0001      000005EA 8F  CO 1216      .10     ADDL  #MAX_BUF$SIZ_UV1,R2      ;
;RNG0001      E2  50  01  F2 121E      .11     AOBLS #MAX_C_XMTUV1,R0,130$ ; Loop if more
-11           54  56  DO 1222      4076      140$: MOVL  R6,R4      ; Set R4 to CDB address
               1222      4077      POPQ  R6      ; Restore R6, R7
               1225      4078      ;
               1228      4079      ; Setup fork process to start CDB timer
               1228      4080      ;
               1228      4081      ;
               38  BB 1228      4082      PUSHR #^M<R3,R4,R5>      ; Save registers
               03  E2 122A      4083      BBSS  #CDB_SIS_V_TIMER,-      ; Br if timer already going
               24 0246 C4      122C      4084      CDB_B_SIS(R4),150$      ;
               04  88 1230      4085      BISB  #DPT$M_NOUNLOAD,-      ; Do not allow driver to be unloaded
               0000000D'EF  1232      4086      DPT$TAB+DPT$B_FLAGS      ; while the TQE is active
               55  0214 C4  DE 1237      4087      MOVAL CDB_L_TQE(R4),R5      ; Get the TQE address
               00 01312D00 8F  7D 123C      4088      MOVQ  #TQE_DELTA,TQE$Q_DELTA(R5) ; Set the delta time
               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$C_SSREPT,TQE$S_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      ;
;RNG0001      52  0114 38  BA 1254      4095 150$: POPR  #^M<R3,R4,R5>      ; Restore registers
-2           DO 1256      .1      MOVL  CDB_L_CSR(R4),R2      ; Get CSR address
               125B      4098      ;
               125B      4099      ; Master reset device
               125B      4100      ;
               0E A2  02  B0 125B      4101      DSBINT UCBS$B_DIPL(R5)      ; Sync access to UCB
               1262      4102      MOVW  #XQ_CSR_M_RESET,CSR(R2) ; Master Reset device

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-2

```

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 :
1266 .1 TIMEWAIT #1,#XQ_CSR_M_ERR,CSR(R2),W ; Wait for 10 usec, bit should not
128C .2 :
51 24 A5 D0 128C .3 MOVL UCBSL_CRB(R5),R1 ; Get CRB address
51 2C A1 D0 1290 .4 MOVL CRBSL_INTD+VECSL_IDB(R1),R1 ; Get IDB address
51 0B A1 9A 1294 .5 MOVZBL IDBSB_VECTOR(R1),R1 ; Set vector address (divided by 4)
51 51 02 78 1298 .6 ASHL #2,R1,R1 ; Convert to real vector address
OC A2 51 B0 129C .7 MOVW R1,VECTOR(R2) ; Set vector address
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
0251 C4 90 12A5 4113 MOVB PHYADD1(R2),CDB_G_HWA+1(R4) ; ...
04 A2 90 12AB 4114 MOVB PHYADD2(R2),CDB_G_HWA+2(R4) ; ...
0252 C4 90 12AE 4115 MOVB PHYADD3(R2),CDB_G_HWA+3(R4) ; ...
06 A2 90 12B1 4116 MOVB PHYADD4(R2),CDB_G_HWA+4(R4) ; ...
0253 C4 90 12B4 4117 MOVB PHYADD5(R2),CDB_G_HWA+5(R4) ; ...
08 A2 90 12B7 4118 :
0254 C4 90 12BA 4119 :
0A A2 90 12BD 4120 :
0255 C4 12C0 4121 :
12C3 4118 :
12C3 4119 : Set CSR mode and enable receiver
12C3 4120 :
OE A2 02 AA 12C3 4121 BICW #XQ_CSR_M_RESET,CSR(R2) ; Clear the master reset
0100 8F A8 12C7 .1 BISW #XQ_CSR_M_ILOOP,CSR(R2) ; DISABLE LOOPBACK
OE A2 12CB .2
12CD .3
12CD .4
06 0249 C4 E9 12CD .5 ASSUME NMASC_LINCN_NOR EQ 0
0300 8F A8 12D2 .6 ASSUME NMASC_LINCN_LOO EQ 1
OE A2 12D6 .7 BLBC CDB_B_CON(R4),160$ ; Br if LOOPBACK is disabled
0040 8F A8 12D8 4127 160$: BISW #XQ_CSR_M_ILOOP!XQ_CSR_M_ELOOP,- ; Else, set internal extended
OE A2 12DC 4127 160$: BISW #XQ_CSR_M_INTENA,CSR(R2); LOOPBACK
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 MOVW 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 MOVW 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 MOVW #XQ_FC_V_INIT,- ; Set function request
20 A2 12FD 4143 CXBSB_XQ_FUNC(R2)
9E 16 12FF 4144 JSB @ (SP)+ ; Complete function request
1301 4145 SETBIT #CDB_STS_V_RUN,- ; Indicate QNA is running

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-5

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-4

09 0249 C4 E8
50 0114 C4 D0
OE A0 01 A8
0116 30
50 01 9A
05

0A A5 0F 90
0B A5 06 90
32 AF 9F
0000000 GF 17

```

1301 4146          CDB_B_STS(R4)          ;
1307 4147          SETBIT #UCBSQ_XQ_RUN,-   ; Indicate UNIT is running
1307 4148          UCBSW_DEVSTS(R5)        ;
130C .1
130C .2          ASSUME NMA$C_LINCN_NOR EQ 0
130C .3          ASSUME NMA$C_LINCN_LOO EQ 1
130C .4          BLBS CDB_B_CON(R4),170$    ; Br if LOOPBACK is enabled, don't
1311 .5          ; enable receive interrupts
1311 .6          MOVL CDB_L_CSR(R4),R0      ; Else, get CSR address
1316 .7          BISW #XQ_CSR_M_RCVENA,CSR(R0); Enable receive interrupts
131A .8 170$:     BSBW FILRCVLIST          ; Start the receives
131D 4153         MOVZBL S^#SS$_NORMAL,R0  ; Return success
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 FOPK_TIMER:
1321 4175         MOVB #DYN$C_TQE,FKBSB_TYPE(R5); Create fork process to start timer
1325 4176         MOVB #IPL$_QUEUEAST,FRBSB_FIPL(R5); Set structure type
1329 4177         PUSHAB B^START_TIMER      ; Push address of fork process
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$L_RQPID(R5) = TQE request type (SS$NGL or SS$REPT)
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
OC A5 10 A5 D0 1338 4205 MOVL FKB$$_FR3(R5),TQE$$_FPC(R5) ; Set address of timer wakeup
      2C A5 90 133D 4206 MOVB TQE$$_RQPID(R5),- ; Set TQE request type
      0B A5
      50 20 A5 7D 1340 4207 TQE$$_RQTYPE(R5) ;
00000000'GF C0 1342 4208 MOVQ TQE$$_DELTA(R5),R0 ; Get delta time
      50 ADDL G^EXE$GQ_SYSTIME,R0 ; Add in current time
00000004'GF D8 134D 4210 ADWC G^EXE$GQ_SYSTIME+4,R1 ; ...
      51
00000000'GF 16 1354 4211 JSB G^EXE$INSTIMQ ; Insert element on timer queue
      135A 4212 ENBINT ; Restore IPL
05 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:
135E 4239     MOVZWL #SS$_INSMEM,R0      ; Setup error
135E 4240     RSB                          ; Set error return
1363 4241                          ; Return to caller
1364 4242
1364 4243 SETUP_MODE:
1364 4244     PUSHL R3                      ; Save R3
1366 4245     MOVZWL #CXBS$HEADER+INIT_C_BUF$SIZE,R1 ; Size of init buffer
136B 4246     JSB G^EXE$ALONONPAGED      ; Allocate the SETUP Transmit buffer
1371 4247     POPL R3                      ; Restore R3
1374 4248     BLBC R0,SETUP_ERR           ; Exit if error
1377 4249     PUSHR #^M<R2,R3,R4,R5>      ; Save registers
1379 4250     MOVCS #0,(R2),#-1,R1,(R2) ; Fill structure with BROADCAST!
137C
137F
1380 4251     POPR #^M<R2,R3,R4,R5>        ; Restore registers
1382 4252
1382 4253     ; Initialize buffer to look like a CXB
1382 4254
1382 4255     MOVL R2,IRP$L XQ SETUP(R3) ; Save address of setup buffer
1387 4256     ASSUME CXBSB_TYPE EQ CXBSW_SIZE+2
1387 4257     ASSUME CXBSB_CODE EQ CXBSB_TYPE+1
1388 4258     MOVL #<DYN$C CXB@16>!- ; Set size and type of structure
1388 4259     CXBS$HEADER+INIT_C_BUF$SIZE,-
138D     CXBSW_SIZE(R2)
138F 4260     MOVW #INIT_C_BUF$SIZE,CXBSW_BCNT(R2) ; Set size of transfer
1393
1395 4261
1395 4262     ; Initialize QNA mode word
1395 4263
1395 4264     ASSUME NMASC_STATE_ON EQ 0
1395 4265     ASSUME NMASC_STATE_OFF EQ 1
1395 4266     CLRW CDB_W_MODE(R4) ; Init mode word
1399 4267     BLBS CDB_B_PRM(R4),10$ ; Br if promiscuous state is OFF

```



```

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

```

XQDRIVER
V04-001

- VAX/VMS QNA driver

L 14

8-JAN-1985 17:49:06

VAX/VMS Macro V04-00

Page 94

SETUP_MODE - SETUP THE TRANSMIT BUFFER T

5-SEP-1984 00:20:54

[DRIVER.BUGSRC]XQDRIVER.MAR;1

(36)

		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::
1433 4354 CLRL R2 ; Fill receive buffer list
1435 4355 ; No buffer here
1435 4356 ADDRCLVLIST::
1435 4357 PUSH R3,R4,R5 ; Add a buffer to the receive list
1437 4358 MOVL CDB_L_UCB0(R4),R5 ; Save registers
143C .1 BBC #CDB_STS_V_RUN,- ; Get UCB address of unit #0
143E .2 10$: CMPW CDB_B_STS(R4),40$ ; Br if device not running
1442 .3 10$: CMPW CDB_W_BSZ(R4),- ; Can new block be allocated?
1446 .4 CDB_W_QUOTA(R4)
1449 .5 BGTRU 35$ ; If GTRU then no, stop loop
144B .6 TSTL R2 ; Buffer already allocated?
144D .7 BNEQ 20$ ; Br if so
144F .8 CLRL R1 ; Zero size
1451 .9 ADDW3 #CXBS$ HEADER+- ; Determine block size needed
1452 .10 CXBS$ TRAILER,-
1452 .11 CDB_W_BSZ(R4),R1
1455
1459 .12 JSB G*EXESALONONPAGED ; Allocate the memory
145F .13 BLBC R0,30$ ; If failure then done
1462 .14 MOVW R1,CXBS$ SIZE(R2) ; Insert size
1466 .15 20$: SUBW CDB_W_BSZ(R4),- ; Subtract from quota
146A .16 CDB_W_QUOTA(R4)
146D 4375 25$: MOVB S*#DYN$C CXB,- ; Insert type
146E 4376 CXBS$ TYPE(R2)
1471 4377 MOVB #XQ_FC_V_RECV,CXBS$ XQ_FUNC(R2) ; Set function request
1475 4378 INSQUE (R2),CDB_Q_RCVBUF(R4) ; Insert block on list
147A 4379 CLRL R2 ; No more buffers given
147C 4380 BRB 10$ ; Continue
147E 4381 :
147E 4382 : Buffer allocation failure
147E 4383 :
147E 4384 30$: SETBIT #XMSV_STS_BUFFAIL,- ; Set buffer alloc failure

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-16

```

52 D4
55 0110 C4 D0
5D 0246 C4 B1
0108 C4
010A C4
38 BB
01 E1
51 0108 C4 B1
004C 8F
51 0108 C4
00000000 GF 16
1C 50 E9
08 A2 51 B0
0108 C4 A2
010A C4
1B 90
0A A2
20 A2 02 90
00EC C4 62 0E
52 D4
C4 11

```

:RNG0001
-1

			147E	4385		CDB_L_DEVDEPEND(R4)	:	
	24	11	1484	4386		50\$:	And give any receives to device
			1486	4387			:	
			1486	4388	35\$:	CLRBIT	:	Clear buffer alloc failure
			1486	4389		CDB_L_DEVDEPEND(R4)	:	
	52	D5	148C	4390		TSTL	:	Any buffer?
	1A	13	148E	4391		BEQL	:	Br if not
02EE	C4	91	1490	4392		CMPB	:	Can we use the additional quota?
02EF	C4		1494	4393		CDB_B_AQUOTA(R4),-	:	
			1497	4394		CDB_B_MQUOTA(R4)	:	
	06	1E	1497	4394		BGEQU	:	Br if not
02EE	C4	96	1499	4395		INCB	:	Else, increment the additional quota
	CE	11	149D	4396		BRB	:	Use buffer, but don't let
			149F	4397			:	QUOTA go negative
			149F	4398			:	
	50	52	149F	4399	40\$:	MOVL	:	Get address of buffer
		06	14A2	4400		BEQL	:	Br if none
00000000	'GF	16	14A4	4401		JSB	:	Deallocate buffer
			14AA	4402			:	
	03	10	14AA	4403	50\$:	BSBB	:	Start the receives
	38	BA	14AC	4404		POPR	:	Restore registers
		05	14AE	4405		RSB	:	
			14AF	4406		.DSABL	:	LSB

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001

-9

```

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 attempts to start any receives that may be pending. This
14AF 4415 : involves dequeuing 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::
14AF 4433 PUSHQ R6 ; Start receive operation
14AF 4434 BBC #CDB_STS_V_RUN,- ; Save R6, R7
14AF 4435 CDB_B_STS(R4),10$ ; Br if device is not running
14AF 4436 :
14AF 4437 : For MicroVAX I, we will not use map registers.
14AF 4438 :
14AF 4439 CPUDISP <<790,5$>,-
14AF 4440 <780,5$>,-
14AF 4441 <750,5$>,-
14AF 4442 <730,5$>,-
14AF 4443 <UV2,5$>,-
14AF 4444 <UV1,40$> ; For MicroVAX I, use alternate path
14AF 4445 :
14AF 4446 5$: ; All but MicroVAX I
14AF 4447 FFC #0,#MAX_C_RCV-1,CDB_B_RCVMAP(R4),R7 ; Get a free mapping slot
14AF 4448 :
14AF 4449 BEQL 10$ ; Br if none - just exit
14AF 4450 REMQUE @CDB_Q_RCVBUF(R4),R3 ; Get a free buffer
14AF 4451 BVC 20$ ; Br if buffer found
14AF 4452 POPQ R6 ; Restore R6, R7
14AF 4453 RSB ; Return to caller
14AF 4454 :
14AF 4455 : Mark slot in use and create buffer address/character count image
14AF 4456 : in receive buffer and load UNIBUS adapter map registers.
14AF 4457 20$: SETBIT R7,CDB_B_RCVMAP(R4) ; Mark slot in use
14AF 4458 MOVB R7,CXB$B-XQ_SLOT(R3) ; Save mapping slot index
14AF 4459 MOVL UCBS$L_CRB(R5),R2 ; Get CRB address
14AF 4460 :
14AF 4461 : Find next ring entry and insert data
14AF 4462 :

```

2B 0246 C4 E1

57 07 00 EB
1A A4
07 13
53 00EC D4 OF
04 1C

22 A3 57 90
52 24 A5 D0

```

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 MOVWB R6,CXB$B_XQ_RING(R3) ; Save ring entry number
56 7C A446 D0 1504 4466 MOVL CDB_L_RRINGVA(R4)[R6],R6 ; Get virtual address of ring entry
    66 B4 1509 4467 CLRW RCV_W_FLAG(R6) ; Zero the FLAG word
    8000 8F B0 150B 4468 MOVW #RCV_STS_M_LAST,- ; Init the STATUS word
    08 A6 150F 4469 RCV_Q_STS(R6) ;
    1511 4470 PUSHQ R3 ; Save IRP, CDB address
    1514 4471 ;
    1514 4472 ; The QNA's receive buffer size must be a multiple of 2
    1514 4473 ;
;RNG0001 50 D4 1514 .1 CLRL R0 ; Zero out high part of R0 longword
    1516 4474 ASSUME <XQ_C_HEADER+XQ_C_CNTRSZ & 1 > EQ 0
    1516 4475 ADDW3 #XQ_C_HEADER+XQ_C_CNTRSZ,- ; Calculate message length
50 0108 C4 A1 1518 4476 CDB_W_BSZ(R4),R0 ;
50 FF 8F 78 151C 4477 ASHL #-1,R0,R0 ; Convert byte count to WORD count
    50 1520 ;
06 A6 50 AE 1521 4478 MNEGW R0,RCV_W_LEN(R6) ; Store length (2's complement)
51 38 A3 9E 1525 4479 MOVAB CXBST R_DATA(R3),R1 ; Get receive buffer data addr
04 A6 51 B0 1529 4480 MOVW R1,RCV_Q_ADDR(R6) ; Set BA0-B48
57 1C A447 DE 152D 4481 MOVAL CDB_L_RCVMAP(R4)[R7],R7 ; Get mapping info slot address
07 09 67 F0 1532 4482 INSV (R7),#9,#7,RCV_W_ADDR(R6) ; Set BA9-BA15 from map reg
    04 A6 1536 ;
67 06 07 EF 1538 4483 EXTZV #7,#6,(R7),R0 ; Get BA16-BA21 also
    50 153C ;
02 A6 50 B0 153D 4484 MOVW R0,RCV_W_ADDRHI(R6) ; Set BA16-BA21 & zero descriptor bits
0A A6 01 B0 1541 4485 MOVW #1,RCV_W_LEN(R6) ; Set low byte not equal to high byte
    54 D4 1545 4486 CLRL R4 ; Use direct data path for rcvs
    1547 4487 ASSUME VEC$B_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
    1557 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 0074 30 1561 .4 BSBW LOAD_RECV ; Give (new) ring address to QNA
;RNG0001 FF6D 31 1564 .12 BRW 5$ ; Let's try it again
;RNG0001 1567 .13
;RNG0001 1567 .14
;RNG0001 1567 .15 40$: ;*****
;RNG0001 1567 .16 ; For MicroVAX I, only.
;RNG0001 1567 .17 ;*****
-15 1567 4510
    1567 4511 ASSUME MAX_C_RCV LE 8
    1567 4512 ASSUME MAX_C_RCVUV1 LT MAX_C_RCV
    1567 4513 FFC #0,#MAX_C_RCVUV1,CDB_B_RCVMAP(R4),R7 ; Get a free mapping slot
57 05 00 EB 1567 4513
    156A ;
    13 156D 4514 BEQL 50$ ; Br if none - just exit
53 00EC D4 OF 156F 4515 REMQUE @CDB_Q_RCVBUF(R4),R3 ; Get a free buffer
    04 1C 1574 4516 BVC 60$ ; Br if buffer found

```

```

05 1576 4517 50$: POPQ R6 ; Restore R6, R7
    1579 4518 RSB ; Return to caller
    157A 4519 :
    157A 4520 : Mark slot in use and create buffer address/character count image
    157A 4521 : in receive buffer and load UNIBUS adapter map registers.
    157A 4522 :
    22 A3 57 90 157A 4523 60$: SETBIT R7,CDB_B_RCVMAP(R4) ; Mark slot in use
    157F 4524 MOVW R7,CXB$B_XQ_SLOT(R3) ; Save mapping slot index
    1583 4525 :
    1583 4526 : Find next ring entry and insert data
    1583 4527 :
    56 19 A4 9A 1583 4528 MOVZBL CDB_B_NEXTRCV(R4),R6 ; Get next ring entry
    19 A4 96 1587 4529 INCB CDB_B_NEXTRCV(R4) ; Bump ring pointer
    F8 8F 8A 158A 4530 BICB #^CZMAX C RCV-1>,- ; Modulo rcv ring size
    19 A4 158D 4531 CDB_B_NEXTRCV(R4) ;
    23 A3 56 90 158F 4532 MOVW R6,CXB$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_Q_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 CLRW R0 ; Zero out high part of R0 longword
    15A2 4540 ASSUME <XQ_C_HEADER+XQ_C_CNTRSZ & 1 > EQ 0 ;
    50 0108 C4 A1 15A2 4541 ADDW3 #XQ_C_HEADER+XQ_C_CNTRSZ,- ; Calculate message length
    50 FF 8F 78 15A4 4542 CDB_W_BSZ(R4),R0 ;
    50 15A8 4543 ASHL #-1,R0,R0 ; Convert byte count to WORD count
    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
    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_LEN(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_VALID,- ; 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 30 15D2 .10 BSBW LOAD_RECVC ; Give (new) ring address to QNA
:RNG00C1 15D5 .12 :
:RNG0001 FF8F 31 15D5 .13 90$: BRW 40$ ; Let's try it again

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-1

```

15D8 .35 .SBTTL LOAD_RECV - LOAD CSR'S WITH RECEIVE RING ADDRESS IF NEEDED
15D8 .36 :++
15D8 .37 : LOAD_RECV - LOAD CSR'S WITH RECEIVE RING ADDRESS IF NEEDED
15D8 .38 :
15D8 .39 : Functional description:
15D8 .40 :
15D8 .41 : This routine loads the receive ring address, if the receive ring has gone
15D8 .42 : invalid.
15D8 .43 :
15D8 .44 : Inputs:
15D8 .45 :
15D8 .46 : R3 = CXB address for receive
15D8 .47 : R4 = CDB address
15D8 .48 : R5 = UCB address
15D8 .49 :
15D8 .50 : IPL = FIPL
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 :--
15D8 .1 LOAD_RECV:: ; Load port command
15D8 .2 MOVL CDB_L_CSR(R4),R2 ; Get CSR address
15DD .3 :
15DD .4 : If the QNA has invalidated the RECEIVE RING, then we must reset the
15DD .5 : starting address of the ring list to point to the current entry.
15DD .6 :
15DD .11 BITW #XQ_CSR_M_RCVINV,CSR(R2); Is receive ring still valid?
15E1 .12 BEQL 40$ ; Br if yes
15E3 .14 MOVZBL CXB$B_XQ_RING(R3),R1 ; Else, get ring entry number
15E7 .15 MOVL CDB_L_RRINGPA(R4)[R1],R1 ; Get the buffer mapping value
15EC .16 MOVW R1,RCVLIST(R2) ; Set address of receive list entry
15F0 .17 ASHL #-16,R1,R1 ; Shift down high order address bits
15F4 :
15F5 .18 MOVW R1,RCVLIST1(R2)
15F9 .19 40$: INSQUE (R3),@CDB_Q_RCVPNPND+4(R4) ; Insert CXB on WAITING queue
15FE 4644 RSB ; Return to caller
15FF 4645

```

```

:RNG0001
:RNG0001 52 0114 C4 D0
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001 0E A2 20 B3
:RNC0001 16 13
:RNG0001 51 23 A3 9A
:RNG0001 51 44 A441 D0
:RNG0001 04 A2 51 B0
:RNG0001 51 F0 8F 78
15F4
:RNG0001 06 A2 51 90
:RNG0001 00F8 D4 63 0E
-58 05 15FE 4644
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 !C(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      :--

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
-5

```

55 54 9E D0 15FF 4673 QNA_INTR::          : DEQNA done interrupt
      18 A4 D0 15FF 4674          : Get IDB address
      18 A4 D0 1602 4675          : Get first UCB address
      52 64 D0 1606 4676          :
      00C8 C5 D0 1606 4677          : Get CSR address
      2A 13 D0 1609 .1             : Get CDB address
      2A 13 D0 160E .2             : Br if CDB not allocated
      25 0246 C4 E9 1610 .3         :
      04 E0 1610 .4             : ASSUME CDB_STS V INITED EQ 0
      1F 0246 C4 E0 1615 4683        : BLBC CDB_B_STS(R4),INTEXIT : Br if NOT inited
      53 0E A2 3C 1617 4684        : BBS #CDB_STS V ERR,-      : Br if there was an error
      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?
      04 13 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.
      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          :
      50 8E 7D 163A 4703 INTEXIT: : Exit interrupt
      50 8E 7D 163A 4704          : MOVQ (SP)+,R0          : Restore Regs

```

XQDRIVER
V04-001

- VAX/VMS QNA driver
QNA_INTR - QNA INTERRUPT SERVICE ROUTINE

G 15

8-JAN-1985 17:49:06
5-SEP-1984 00:20:54

VAX/VMS Macro V04-00
[DRIVER.BUGSRC]XQDRIVER.MAR;1

Page 102
(41)

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      :
1644 4737      :     CDB_L_LASTCSR(R4) = new CSR contents
1644 4738      :
1644 4739      :--
1644 4740
1644 4741 SCHED_FORKC::          ; Schedule fork process, clr R3
1644 4742      CLRL      R3          ; No device error
1644 4743 SCHED_FORK::          ; Schedule fork process
1644 4744      BBSS      #CDB_STS_V_FORK_PEND,CDB_B_STS(R4),10$ ; Br if fork pending
1644 4745
1644 4745      ASSUME   CDB_L_FQFL EQ 0
1644 4746      MOVL    R4,R5          ; Get CDB fork block address
1644 4747      PUSHAB  B^FORK PROC    ; Else, set address of fork process
1644 4748      JMP     G^EXES$FORK    ; Schedule the fork and return
1644 4749
1644 4750 10$:      BBC      #XQ_CSR_V_ERR,R3,20$ ; Br if not error
1644 4751      MOVL    R3,CDB_C_LASTCSR(R4) ; Save last CSR value
1644 4752 20$:      RSB          ; Return to caller

```

:RNG0001
-1

```

          53  D4
0246 C4  02  E2
          OC
          55  54  D0
          63'AF 9F
00000000'GF 17

```

:RNG0001
-1

```

04 53  0E  E1
10 A4  53  D0
          05  1660

```

```

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 OE E1 1669 4779 BBC #XQ_CSR_V_ERR,R3,10$ ; Br if not an error
010F C4 53 90 166D 4780 SETBIT #XMSV_ERR_FATAL,CDB_L_DEVDEPEND(R4) ; Indicate fatal error
55 0110 C4 D0 1673 4781 MOVB R3,CDB_L_DEVDEPEND+3(R4) ; Save low byte of CSR
0082 C5 B6 1678 4782 MOVL CDB_L_UCB0(R4),R5 ; Get UCB #0 address
0889 31 167D 4783 INCL UCB$W_ERRCNT(R5) ; Bump error counter
1681 4784 BRW SHUTDOWN_QNA ; Shutdown the QNA device
1684 4785
1684 4786 3$: ;
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
168A .1 7$:
014B 31 168A .8 8$: BRW 60$ ; Complete transmits
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 none
50 1A A2 3C 1696 4800 INCC CDB_L_DBRCTR(R4) ; Count blocks received
16A0 4801 MOVZWL CXB$W_BCNT(R2),R0 ; Get byte count
16A4 4802 CNTR R0,CDB_L_BRCCTR(R4),L ; Count bytes received
16B0 4803 BLBC CXB$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
B5 14 OE E0 16CA 4806 17$: BBS #RCV_STS_V_ERR,- ; Br if FATAL receive error
0D 31 16CC 4807 ;
16CF 4808 BBC #RCV_STS_V_ESETUP,- ; Br if NOT an ESETUP receive

```

:RNG0001
:RNG0001
:RNG0001
-1

```

06 14 A2      16D1 4809      CXBSW_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      ;
51 44 A2      3C 16DA 4812 20$: MOVZWL CXBSW_R_PTYPE(R2),R1 ; Pick up protocol type from buffer
      16DE 4813
      16DE 4821
0260 8F 51    B1 16DE 4822 21$: 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      ;
      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$: BSBW MATCH_PROTYP ; Try to match protocol type
      1F 50 E8 16F3 4830      BLBS RO,27$ ; 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
      10 A2 DD 1707 4834 25$: PUSHL CXBSL_LINK(R2) ; Else, no such protocol type
      FD28 30 170A 4835      BSBW ADDRCLIST ; Save next in chain
      52 8ED0 170D 4836      POPL R2 ; Add buffer to receive list
      F5 12 1710 4837      BNEQ 25$ ; Restore next buffer
      FF7B 31 1712 4838      BRW 15$ ; Loop if more
      1715 4839 ; Look for next completed buffer
      1715 4840 ;
      1715 4841 ; If there is a promiscuous user, then copy the packet for the promiscuous
      1715 4842 ; user. There is a chance that the data received is not for the protocol
      1715 4843 ; type user just found, we will have to re-verify that the destination
      1715 4844 ; address is for our physical address. This is because if we are running
      1715 4845 ; promiscuous mode, then we will receive all packets, including those not
      1715 4846 ; intended for the protocol user.
      1715 4847 27$: ASSUME NMASC_STATE_ON EQ 0
      1715 4848      ASSUME NMASC_STATE_OFF EQ 1
      23 0247 C4 E8 1715 4849      BLBS CDB_B_PRM(R4),32$ ; Br if hardware is NOT in promiscuous mode
      10 38 A2 E8 171A 4850      BLBS CXBSG_R_DEST(R2),30$ ; Br if multicast address, this
      171E 4851 ; will be checked later.
      38 A2 D1 171E 4852      CMPL CXBSG_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 CXBSG_R_DEST+4(R2),- ; Are we sure?
      025A C4 1729 4856      CDB_G_PHYADR+4(R4) ;
      CB 12 172C 4857      BNEQ 23$ ; Br if no, don't copy packet
      172E 4858 ;
      172E 4859 ; Copy the packet for the promiscuous user
      172E 4860 ;
      55 0210 C4 DD 172E 4861 30$: PUSHL R5 ; Save user's UCB address
      03 13 1735 4862      MOVL CDB_L_PRMUSER(R4),R5 ; Get PROMISCUOUS user's UCB address
      0285 30 1737 4863      BEQL 31$ ; Br if none
      55 8ED0 173A 4864      BSBW COPY_RCV ; Give buffer to promiscuous user
      173D 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 CXBSG_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 RO,34$ ; Br if success

```

```

1747 4873 33$: INCC UCBSW_XQ_MNECTR(R5),W ; Else, multicast not enabled
1751 4874 INCC CDB_W_UFDCTR(R4),W ; Also added in unrecognized frame dest
AA 11 175B 4875 BRB 25$ ; 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$: TSTL CXBSL_LINK(R2) ; Is this a chained message?
05 13 1760 4881 BEQL 36$ ; Br if not
1762 4882 ASSUME NMASC_STATE_ON EQ 0
1762 4883 ASSUME NMASC_STATE_OFF EQ 1
4A 00E0 C5 E8 1762 4884 BLBS UCBSB_XQ_DCR(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$: MOVZWL UCBSW_DEVBUFSIZ(R5),R0 ; Get size of user's max buffer
176B 4890 :&&
176B 4891 :&& the following code could cause problems for an altstart user,
176B 4892 :&& if the altstart user happens to receive a buffer which is 1 or 2
176B 4893 :&& bytes longer than they are capable of handling. Only if the
176B 4894 :&& protocol is not "padded", because the size check allows for
176B 4895 :&& 2 bytes of count to be subtracted from the message size.
176B 4896 :&&
176B 4897 :
176B 4898 : Check the size of the received buffer against what the user protocol can
176B 4899 : handle.
176B 4900 :
51 02 A3 176B 4901 SUBW3 #XQ_C_CNTRSZ,- ; Get the size of the receive buffer
1A A2 176D 4902 ; minus the count word
50 51 B1 1770 4903 CMPW R1,R0 ; Is the received size larger than
1773 4904 ; what the user can handle?
0192 3C 1A 1773 4905 BGTRU 45$ ; Br if yes, error
C5 D5 1775 4906 TSTL UCBSL_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$: MOVAB UCBSQ_XQ_RCVREQ(R5),R1 ; Assume that we are running in
1786 4912 ; exclusive mode
09 68 A5 E1 1786 4913 BBC #UCBSV_XQ_SHARE,- ; Br if UCB is NOT in SHARED mode
1788 4914 UCBSW_DEVSTS(R5),38$ ;
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$: REMQUE @ (RT),R3 ; Remove waiting IRP
0B 1D 1798 4927 BVS 40$ ; Br if none - queue for later
00CC C5 50 A2 179A 4928 SUBW R0,UCBSW_XQ_QUOTA(R5) ; Else, lessen quota so it can be
179F 4929 ; ..increased on completion

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
-1

```

02E8 30 179F 4930      BSBW  FINISH_RCV_IO      ; And finish the I/O
FEED 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$: SUBW  RO_UCBSW_XQ_QUOTA(R5) ; Decrement the quota
      1E 17AA 4936      BGEQU  50$                ; Br if we can buffer request
00CC C5 50 A0 17AC 4937      ADDW  RO_UCBSW_XQ_QUOTA(R5) ; Replace quota
      17B1 4938      45$: INCC  UCBSW_XQ_UBOCTR(R5),W ; Else, no buffer available
      17BB 4939      INCC  CDB_W_UBOCTR(R4),W ; ...don't forget CDB counter
      FF3F 31 17C5 4940      BRW   25$                ; Return buffer
      17C8 4941      ;
      17C8 4942      50$: ASSUME UCBSQ_XQ_RCVREQ EQ UCBSQ_XQ_RCVMSG+8
      17C8 4943      ASSUME SHR_Q_RCVREQ EQ SHR_Q_RCVMSG+8
      51 04 C2 17C8 4944      SUBL  #4,R1 ; Backup to backward link pointer
      17CB 4945      ; of the message queue
      00 B1 62 0E 17CB 4946      INSQUE (R2),@(R1) ; Queue received msg for later
      FC61 30 17CF 4947      BSBW  FILLRCVLIST ; Try to fill the receive list
      1291 30 17D2 4948      BSBW  POKE_USER ; Deliver ASTs
      FEB8 31 17D5 4949      BRW   15$                ; 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
      012B 31 17DE .1 ; BRW 190$ ; Else, all done
      17E1 .2 ;
      17E1 .3 65$: BRW 190$ ; All done
      0128 31 17E1 .7 ;
      17E4 4956 ;
      56 0105 C4 9A 17E4 4957 70$: MOVZBL CDB_B_LASTXMT(R4),R6 ; Get last ring entry completed
      56 009C C446 D0 17E9 4958      MOVL  CDB_L_XRINGVA(R4)[R6],R6 ; Get address of last ring entry
      OF E1 17EF 4959      BBC   #XMT_STS_V_LAST,- ; Br if done
      05 08 A6 17F1 4960      XMT_W_STS(R6),75$ ;
      OE E0 17F4 4961      BBS   #XMT_STS_V_ERR,- ; Br if not done
      EB 08 A6 17F6 4962      XMT_W_STS(R6),65$ ; ..leave
      OA 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 OF 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$: DECB  CDB_B_XMTCNT(R4) ; One less transmit pending
      0105 C4 96 180F 4970      INCB  CDB_B_LASTXMT(R4) ; Bump ring pointer
      FC 8F 8A 1813 4971      BICB  #^C<MAX C XMT-1>,- ; Modulo receive ring entry size
      0105 C4 1816 4972      CDB_B_LASTXMT(R4) ;
      1819 4973      ;
      1819 4974      ; Transmit complete
      1819 4975      ;
      1819 4976      ;
      1819 4977      CLRBIT XMT_DSC_V_VALID,- ; Indicate that buffer is not valid
      01 A6 90 181E 4978      MOVW  XMT_Q_ADDRHI(R6) ;
      0119 C4 1821 4979      MOVW  XMT_Q_FLAG+1(R6),- ; Save diagnostic return info
      08 A6 B0 1824 4980      MOVW  CDB_B_DIAG1(R4) ;
      011A C4 1827 4981      MOVW  XMT_Q_STS(R6),- ; Save diagnostic return info
      55 24 A3 D0 182A 4982      MOVL  CDB_Q_DIAG2(R4) ;
      182E 4983      MOVL  CXB$C_T_UCB(R3),R5 ; Get (presumed) UCB address
      ASSUME CXB$C_T_UCB EQ CXB$L_T_IRP

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-3

```

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$>,-
183F .1 <UV2,90$>,-
183F .2 <UV1,100$>> ; Skip map registers if MicroVAX I
185B .3
185B .4 90$: ; All but MicroVAX I
185B .5
51 24 A5 D0 185B .6 MOVL UCBS$L_CRB(R5),R1 ; Get CRB address
185F 4997 ASSUME VEC$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$W_MAPREG(R1) ; and data path number
52 95 1865 5001 TSTB R2 ; Was it the pre-allocated one?
OD 13 1867 5002 BEQL 95$ ; 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$W_MAPREG+3
50 3B A4 94 1876 5008 CLRB CDB_L_XMTMAP+3(R4) ; Clear data path number used
1A A3 3C 1879 5009 100$: MOVZWL CXB$W_BCNT(R3),R0 ; Get byte count of message
51 50 D0 187D 5010 MOVL R0,R1 ; Copy length for accounting
1880 5011
1880 5012 ; Perform accounting for the QNA
1880 5013
1880 5014
188A 5015 INCC CDB_L_DBSCTR(R4) ; Count blocks sent
50 50 10 78 1896 5016 CNTR R1,CDB_L_BSNCTR(R4),L ; Count bytes sent
50 01 B0 189A 5017 ASHL #16,R0,R0 ; Move to high word
OC E4 189D 5018 MOVW S^#SS$ NORMAL,R0 ; Set completion status
42 02 A6 189F 5019 BBSC #XMT_DSC_V_SETUP,- ; Br if SETUP operation
05 08 A6 OE 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$: CNTR R1,UCBS$L_XQ_SBYCTR(R5),L ; Bump the bytes sent counter
18B8 5027 INCC UCBS$L_XQ_SBCCTR(R5),L ; Bump the blocks sent counter
08 A6 04 EF 18C2 5028 EXTZV #XMT_STS_V_COL,- ; Get number of collisions
51 18C4 5029 #XMT_STS_S_COL,XMT_W_STS(R6),R1
1A 13 18C8 5030 BEQL 120$ ; Br if none
51 97 18CA 5031 DECB R1 ; More than one?
OC 13 18CC 5032 BEQL 115$ ; Br if only one
18CE 5033 INCC CDB_L_BSMCTR(R4) ; Count blocks sent with multiple errors
0A 11 18DB 5034 BRB 120$ ; Continue
18DA 5035 115$: INCC CDB_L_BS1CTR(R4) ; Count blocks sent with 1 error

```


1B 24 A3	E8	18E4	5036		BLBS	CXB\$T_IRP(R3),130\$:	BR if FAST interface CXB
53 24 A3	D0	18E8	5037 120\$:		MOVL	CXB\$T_IRP(R3),R3	:	Else, get IRP address
	DD	18EC	5038		PUSHL	R0	:	Save status return
50 0094 C3	D0	18EE	5039		MOVL	IRP\$L_XQ_SETUP(R3),R0	:	Is there a SETUP mode buffer
	06	18F3	5040		BEQL	125\$:	Br if none
00000000'GF	16	18F5	5041		JSB	G^COM\$DRVDEALMEM	:	Deallocate the buffer
	50 8ED0	18FB	5042		POPL	R0	:	Restore status return
	024F	30	18FE	5043 125\$:	BSBW	IO_DONE	:	Finish the I/O request
	03	11	1901	5044	BRB	170\$:	Continue in common code
	013B	30	1903	5045 130\$:	BSBW	FINISH_XMT_FFI	:	Else, complete FFI XMIT
	EB7E	30	1906	5046 170\$:	BSBW	XMT_ALT_START	:	Start up any other transmits
	FECC	31	1909	5047	BRW	60\$:	Look for next completion
			190C	5048				
			190C	5049				
	05		190F	5050 190\$:	POPQ	R6	:	Restore R6, R7
			1910	5051	RSB			
				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
06 14 A2 E1 191A 5074 BBC     #RCV_STS_V CRCERR,-      ; Br if not a CRC error
191C 5075          CXBSQ R STS(R2),20$
191F 5076 SETBIT #0,CDB_W_RFLMAP(R4) ; Indicate CRC error
06 14 A2 E1 1925 5077 20$: BBC   #RCV_STS_V FRAME,-      ; Br if not a framing error
1927 5078          CXBSQ R STS(R2),40$
192A 5079 SETBIT #1,CDB_W_RFLMAP(R4) ; Indicate FRAME error
06 14 A2 E1 1930 5080 40$: BBC   #RCV_STS_V RUNT,-      ; Br if not a RUNT packet
1932 5081          CXBSQ R STS(R2),90$
1935 5082 SETBIT #2,CDB_W_RFLMAP(R4) ; Indicate FRAME error
05 193B 5083 90$: RSB
193C 5084
193C 5085
193C 5086 :++
193C 5087 : XMT_ERROR - Process transmit errors
193C 5088 :
193C 5089 : Functional description:
193C 5090 :
193C 5091 : This routine adjusts all appropriate counters and checks all errors.
193C 5092 :
193C 5093 : Inputs:
193C 5094 :     R4 = CDB address
193C 5095 :     R6 = Transmit ring entry address
193C 5096 :
193C 5097 : Outputs:
193C 5098 :     R0 = error code
193C 5099 :
193C 5100 :--
193C 5101 :
193C 5102 XMT_ERROR:
50 20C4 8F B0 1946 5103 INCC   CDB W SFLCTR(R4),W      ; Count send failures
0y E1 194B 5104 MOVW  #SS$_COMMHARD,R0      ; Assume No Carrier failure
50 0B 08 A6 B0 194D 5105 BBC     #XMT_STS_V ABORT,-      ; Br if NOT 16 retries failed
0334 8F B0 1950 5106          XMT_Q STS(R6),20$
1955 5107 MOVW  #SS$_DEVREQERR,R0      ; Else, DEVREQERR error
0C E1 195B 5108 20$: SETBIT #0,CDB_W_SFLMAP(R4) ; Set bitmap
0B 08 A6 195D 5109          BBC   #XMT_STS_V LCAR,-      ; Br if NOT Loss of Carrier
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$: BBC #XMT_STS-V FAIL - ; Br if NOT Collision check failure
50 OF 08 A6 196D 5114 XMT # STS(R6),60$
      005C 8F B0 1970 5115 MOVW #SS$ DATACHECK,R0 ; Else, DATACHECK error
      1975 5116 INCC CDB # CDCCTR(R4),W ; Count separate counter
      0B E1 197F 5117 60$: BBC #XMT_STS V NOCAR,- ; Br if NOT Carrier failure
      06 08 A6 1981 5118 XMT # STS(R6),90$
      1984 5119 SETBIT #2,CDB_W_SFLMAP(R4) ; Set bitmap
      05 198A 5120 90$: RSB
```

```

1980 5122 .SBTTL SUBROUTINES TO FIND SHR MATCH ON SOURCE ADDRESS
198B 5123 :++
198B 5124 : Functional description:
198B 5125 :
198B 5126 : Subroutine to find SHR data structure for user
198B 5127 :
198B 5128 : Inputs:
198B 5129 :     R2 = Receive CXB address
198B 5130 :     R5 = UCB address
198B 5131 :
198B 5132 : Outputs:
198B 5133 :     R1 = Address if SHR data structure if match
198B 5134 :     All other registers preserved.
198B 5135 :     Z-Bit set then match.
198B 5136 :     Z-Bit clear then no match.
198B 5137 :--
198B 5138
05 BB 198B 5139 MATCH_SRC: ; Try to find shared user
198B 5140 PUSHF #^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 UCBSQ_XQ_SHARE(R5),R0 ; Save address of listhead
51 51 50 D0 1992 5145 MOVL R0,R1 ; Copy listhead address
51 61 D0 1995 5146 ASSUME SHR_L_QFL EQ 0
50 51 D1 1995 5147 20$: MOVL (R1),R1 ; Get next in list
13 13 199B 5148 CMPL R1,R0 ; Back to start of list?
13 10 199D 5149 BEQL 30$ ; Br if yes - no source match
F4 12 199F 5150 BSBB CHECK_SRC ; Check for match
OC 11 19A1 5151 BNEQ 20$ ; Br if none
19A3 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$: MOVL UCBSL_XQ_DEFUSR(R5),R1 ; Get address of default user
02 13 19A8 5157 BEQL 40$ ; Br if no default user
50 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
12 A1 3E A2 D1 19B2 5177 CHECK_SRC: ; Check for match with SHR data base
19B2 5178 CMPL CXBSG_R_SRC(R2),SHR_G_DEST(R1) ; Source address match?

```

XQDRIVER
V04-001

- VAX/VMS QNA driver
SUBROUTINES TO FIND SHR MATCH ON SOURCE

E 16

8-JAN-1985 17:49:06
5-SEP-1984 00:20:54

VAX/VMS Macro V04-00
[DRIVER.BUGSRC]XQDRIVER.MAR;1

Page 113
(45)

05	12	1987	5179	BNEQ	10\$: Br if no - try for next
42 A2	B1	1989	5180	CMPW	CXB\$G R SRC+4(R2),-		: Really match?
16 A1		19BC	5181		SHR_G_DEST+4(R1)		: Return to caller
	05	19BE	5182	10\$:	RSB		

XQDRIVER
V04-001

- VAX/VMS QNA driver

G 16

8-JAN-1985 17:49:06

VAX/VMS Macro V04-00

Page 115

COPY_RCV - Copy a receive buffer for the

5-SEP-1984 00:20:54

[DRIVER.BUGSRC]XQDRIVER.MAR;1

(46)

```
:RNG0001      1A0B      .9      INCC      UCBSW_XQ_UBUCTR(R5),W      ; Else, no buffer available
:RNG0001      1A15      .10     INCC      CDB_W_UBUCTR(R4),W      ; ...don't forget CDB counter
:RNG0001      10 A2 DD 1A1F      .11 50$:  PUSHL   CXBSL_LINK(R2)      ; Save next in chain
:RNG0001      50 52 D0 1A22      .12     MOVL    R2,R0      ; Copy the buffer address
:RNG0001      00000000'GF 16 1A25      .13     JSB     G^COM$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 OE 1A32      .18 60$:  INSQUE  (R2),@UCBSQ_XQ_RCVMSG+4(R5) ; Queue received msg for later
-19          F9F9 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   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 request*
1A41 5282 PUSHL R4 : Save R4
54 0192 C5 D0 1A43 5283 MOVL UCBSL XQ_FFI(R5),R4 : Get FFI block address
14 B4 16 1A48 5284 JSB @FFI$C_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::
1A4F 5313      PUSHL  R4          ; Save R4
1A4F 5314      MOVL  R2,R3      ; Copy CXB address
1A4F 5315      MOVC  CXB$G_R_SRC(R3),- ; Copy source address
1A4F 5316      MOVC  CXB$Q_STATION(R3),- ;
1A4F 5317      MOVW  #CXB$T_R_USERDAT,CXB$W_BOFF(R3) ; Set offset to received data
1A4F 5318      ASSUME  NMASC_STATE_ON EQ 0
1A4F 5319      ASSUME  NMASC_STATE_OFF EQ 1
1A4F 5320      BLBS  UCB$B_XQ_PAD(R5),30$ ; Br if padding is disabled
1A4F 5321      MOVW  CXB$W_R_SIZE(R3),- ; Else, set real size of buffer
1A4F 5322      CXB$W_BCNT(R3)
1A4F 5323      ADDW  #XQ_C_CNTRSIZE,CXB$W_BOFF(R3) ; Adjust offset
1A4F 5324      MOVL  CDB$L_DEVDEPEND(R4),R0 ; Set controller bits
1A4F 5325      BLSL  UCB$L_DEVDEPEND(R5),R0 ; Set status flags
1A4F 5326      MOVL  UCB$L_XQ_FFI(R5),R4 ; Get FFI block address
1A4F 5327      JSB  @FFI$C_RECV_DONE(R4) ; Call back the user with CXB
1A4F 5328      POPL  R4          ; Restore R4
1A4F 5329      MOVL  R3,R7      ; Was buffer consumed?
1A4F 5330      BEQL  90$       ; Br if YES
1A4F 5331      BSBW  ADDRCLIST ; Else, add buffer to receive list
1A4F 5332      RSB

```

```

53 54 DD 1A4F 5312
   52 DO 1A51 5313
3E A3 7D 1A54 5314
28 A3 7D 1A57 5315
0046 8F B0 1A59 5316
   18 A3 1A5D 5317
09 00DD C5 E8 1A5F 5318
   46 A3 B0 1A64 5319
   1A A3 1A67 5320
18 A3 02 A0 1A69 5321
50 010C C4 D0 1A6D 5322
50 44 A5 C8 1A72 5323
54 0192 C5 D0 1A76 5324
   18 B4 16 1A7B 5325
   54 B4 16 1A7B 5326
   52 53 DO 1A7E 5327
   03 13 1A81 5328
   F9AC 30 1A84 5329
   05 1A86 5330
   05 1A89 5331

```

```

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::
2C A3 52 D0 1A8A 5361      MOVL   R2,IRP$S_SVAPTE(R3)      ; Finish receive I/O request
3E A2 7D 1A8E 5362      MOVQ   CXB$G_R_SRC(R2)      ; Save block address
40 A3 1A91 5363      IRPSQ  STATION(R3)      ; Copy source address for DECnet
27 2A 07 E1 1A93 5364      BBC    #IRP$V_DIAGBUF,-      ; Br if no diagnostic buffer
51 4C A3 DO 1A95 5365      IRP$W_STS(R3),10$      ;
50 0C A1 9E 1A98 5366      MOVL   IRP$S_DIAGBUF(R3),R1      ; Get diagnostic buffer
08 A1 1A B1 1A9C 5367      MOVAB  RHDR T DATA(R1),R0      ; Assume this is just a read header
50 30 A1 13 1AA0 5368      CMPW  #RHDR_C_LENGTH,DIAG_W_SIZE(R1) ; Is this just a header buffer?
0119 C4 10 13 1AA4 5369      BEQL  5$      ; Br if yes
011A C4 9E 1AA6 5370      MOVAB  DIAG T RDATA(R1),R0      ; Else, must be a diagnostic buffer
14 A2 90 1AAA 5371      MOVB  CXB$B_R_FLAGS(R2),-      ; Save diagnostic return info
38 A2 0E 28 1AAD 5372      CDB B DIAG1(R4)      ;
60 3C BB 1AB0 5373      MOVW  CXB$W_R_STS(R2),-      ;
04 A2 3C A3 DO 1AB3 5374      CDB W DIAG2(R4)      ;
51 1A A2 3C 1AB6 5375 5$:  PUSHR  #^M<R2,R3,R4,R5>      ; Save registers
00CC C5 28 1AB8 5376      MOVC3 #RHDR_C_DATA,CXB$T_R_DATA(R2),(R0) ; Move header info
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
42 A5 A0 1AC3 5379      MOVL  IRP$S_XQ_DATBUF(R3),4(R2) ; Set address of user buffer
1A8A 5380      ADDW  UCB$W_DEVBUFSIZ(R5),-      ; Adjust receive buffer quota
1A8A 5381      UCB$W_XQ_QUOTA(R5)      ;
1A8A 5382      MOVZWL CXB$W_BCNT(R2),R1      ; Find length of received message
1A8A 5383      :
1A8A 5384      : Perform accounting on a per protocol type basis.
1A8A 5385      :
1A8A 5386      :     CNTR  R1,UCB$S_XQ_RBYCTR(R5),L ; Bump the bytes received counter
1A8A 5387      :     INCC  UCB$S_XQ_RBCTR(R5),L   ; Bump the blocks received counter
1A8A 5388      :
1A8A 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 UCBSB_XQ_PAD(R5),15$ : Br if padding is disabled
50 46 A2 3C 1AED 5395 : MOVZWL CXBSW_R_SIZE(R2),R0 : Else, pick up real size of
62 02 C0 1AF1 5396 : ADDL #XQ_C_CNTSIZ,(R2) : MSG from the message itself
1AF4 5398 : : Move pointer past the count field
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?
51 32 A3 1B 1B04 5411 : BLEQU 20$ : Br if no - okay
0838 8F 3C 1B06 5412 : MOVZWL IRP$W_BCNT(R3),R1 : Else, set size to minimum of two
3A A3 B0 1B0A 5413 : MOVW #SS$_DATAOVERUN,IRP$W_XQ_STATUS(R3) : Set return status
1B10 5414 :
1B10 5415 : : If chained buffers, then setup IRP$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$: TSTL CXBSL_LINK(R2) : Is this a complex chained buffer?
0B 13 1B13 5420 : BEQL 30$ : Br if no - set transfer size
05 E1 1B15 5421 : RBC #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 : B1SW #IRP$M_COMPLX,- : Else indicate complex
2A A3 1B1C 5424 : IRP$W_STS(R3) : chained buffers
32 A3 51 B0 1B20 5425 : BRB 40$ : And don't change user buffer size
50 51 10 78 1B24 5426 30$: MOVW R1,IRP$W_BCNT(R3) : Set size of transfer
06 12 *B28 5427 40$: ASHL #16,R1,R0 : Set buffer size in status
0054 8F B0 1B2A 5428 : BNEQ 50$ : Br if success
3A A3 1B2E 5429 45$: MOVW #SS$_CTRLERR,- : Set data transfer error
1B30 5430 : IRP$W_XQ_STATUS(R3)
50 3A A3 B0 1B30 5431 :
1A 10 1B34 5432 50$: MOVW IRP$W_XQ_STATUS(R3),R0 : Get status
F8FA 31 1B36 5433 60$: BSBB IO_DONE : Post the I/O request
1B39 5434 : BRW FICLRCVLIST : Fill up the receive buffers
1B39 5435 :
1B39 5436 : : Complete an I/O request packet
1B39 5437 :
50 2C 9A 1B39 5438 ABORT_PKT: : Abort the I/O request
1B3C 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
51 00C8 C5 D0 1B40 .1 : MCVL UCBSL_XQ_CDB(R5),R1 : Get CDB address
010C C1 09 E1 1B45 5445 : BBC #XMSV_STS_TIMO,CDB_L_DEVDEPEND(R1),IO_DONE : Br if not a timeout
05 1B4A :

```

:RNG0001
:RNG0001
:RNG0001

:RNG0001
:RNG0001
:RNG0001
-8

```

50 022C 8F 3C 1B4B 5446 MOVZWL #SS$_TIMEOUT,R0 ; Else, return real error code
      1B50 5447 IO_DONE: ; Complete an I/O request
38 A3 50 D0 1B50 5448 MOVL RO,IRP$L_IOST1(R3) ; Set status return and size
      1B54 5449 IO_DONE1: ; Alternate entry point
55 1C A3 D0 1B54 5450 MOVL IRP$L_UCB(R3),R5 ; Get UCB address
3C A3 44 A5 D0 1B58 5451 MOVL UCBS$L_DEVDEPEND(R5),IRP$L_IOST2(R3) ; Set other info
51 00C8 C5 D0 1B5D .1 MOVL UCBS$L_XQ_CDB(R5),R1 ; Get CDB address
      06 13 1B62 .2 BEQL IO_DONE2 ; Br if no CDB
      010C C1 C8 1B64 .3 BISL CDB_L_DEVDEPEND(R1),IRP$L_IOST2(R3) ; Set controller bits
      3C A3 1B68
      1B6A .4 IO_DONE2: ; P1 IOST2 already setup
      1B6A .5 PUSHL R4 ; Save R4
54 00C8 C5 D0 1B6C .6 MOVL UCBS$L_XQ_CDB(R5),R4 ; Get CDB address
      07 E1 1B71 5460 BBC #IRP$V_DIAGBUF - ; Br if no diagnostic buffer
      22 2A A3 1B73 5461 IRP$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 past start time
00000000'GF 7D 1B84 5466 MOVQ G^EXE$GQ_SYSTIME,(R0)+ ; Insert stop time
      80 1B8A
51 0110 C4 D0 1B8B 5467 MOVL CDB_L_UCB0(R4),R1 ; Get address of UCB #0
80 0082 C1 3C 1B90 5468 MOVZWL UCBS$W_ERRCNT(R1),(R0)+ ; Insert error counter
      0218 30 1B95 5469 BSBW REG_DUMP ; Dump registers
      54 3ED0 1B98 5470 20$: POPL R4 ; Restore R4
00000000'GF 17 1B9B 5471 JMP G^COM$POST ; Post the I/O and return

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-6

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-11

```

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
1BA1 5480      : buffers.
1BA1 5481      :
1BA1 5482      : Inputs:
1BA1 5483      :
1BA1 5484      :     R4 = CDB address
1BA1 5485      :
1BA1 5486      :     IPL = FIPL
1BA1 5487      :
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$V_STS if the receive
1BA1 5500      :     buffer is comprised of complex chained buffers.
1BA1 5501      :--
1BA1 5502      : .ENABL  LSB
1BA1 5503      ASSEM_PKTS::      : Assemble receive packets
1BA1 5504      BSBW  NEXTMSG      : Get first message
1BA1 5505      BLBC  RO,100$      : Br if none
1BA1 5506      :
1BA1 5507      : Save number of messages in chain in CXBSW_R_NCHAIN and total size of
1BA1 5508      : all messages in CXBSW_BCNT
1BA1 5509      :
1BA1 5510      :
1BA1 5511      :     MOVW  #1,CXBSW_R_NCHAIN(R2)      : Compute total number of buffers
1BA1 5512      :     MOVW  CXBSW_LENGTH(R2),R0      : Get size of message
1BA1 5513      :     BBC   #RCV_STS_V_LAST,-      : Br if end of packet
1BA1 5514      :           CXBSW_R_STS(R2),40$      : ..all done with this loop
1BA1 5515      :     CLRW  CXBSW_BCNT(R2)          : Init total size of buffers so far
1BA1 5516      :     PUSHL R2                      : Save first receive buffer address
1BA1 5517      :     MOVZBL #MAX_C_CHAIN,R3        : Allow n messages in chain
1BA1 5518      :     MOVW  RCV_Q_LEN(R6),-        : Set size of buffer to maximum per rcv
1BA1 5519      :           CXBSW_LENGTH(R2)      :
1BA1 5520      :     PUSHL R2                      : Save address of current bufr in chain
1BA1 5521      :     BSBW  NEXTMSG                : Try for next message
1BA1 5522      :     POPL  R1                      : Get address of last buffer in chain
1BA1 5523      :     BLBC  RO,20$                 : Toss all messages on error
1BA1 5524      :     MOVL  (SP),R0                 : Get address of first in chain
1BA1 5525      :     MOVL  R2,CXBSL_LINK(R1)       : Store address in chain
1BA1 5526      :     ADDW  CXBSW_LENGTH(R1),-      : Compute total size of all buffers
1BA1 5527      :           CXBSW_BCNT(R0)         : in chain - so far
1BA1 5528      :     INCW  CXBSW_R_NCHAIN(R0)     : Compute number of msgs in chain
1BA1 5529      :     BBC   #RCV_STS_V_LAST,-      : Br if end of packet
1BA1 5530      :           CXBSW_R_STS(R2),30$    :
1BA1 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
OB A1 90 1BF5 5543 : MOVB CXBSB R_FLAGS(R1),- ; Save only last message buffer info
OB A2 1BF8 5544 : CXBSB R_FLAGS(R2) ; in first message of chain
14 A1 B0 1BFA 5545 : MOVW CXBSW R_STS(R1),- ; DITTO
14 A2 1BFD 5546 : CXBSW R_STS(R2)
50 OC A1 B0 1BFF 5547 : MOVW CXBSW_LENGTH(R1),R0 ; Save size of entire message
1A A2 A2 1C03 5548 : SUBW CXBSW_BCNT(R2),- ; Compute size of last message
OC A1 1C06 5549 : CXBSW_LENGTH(R1) ; and store in CXB format
1A A2 50 B0 1C08 5550 40$: MOVW R0,CXBSW_BCNT(R2) ; Return size of complete message
50 01 9A 1C0C 5551 50$: 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
10 A2 DD 1C13 5557 TOSSMSG:INCC CDB_W_LBECTR(R4),W ; Up the counter
F812 30 1C1D 5558 110$: PUSHL CXBSL_LINK(R2) ; Save address of next in chain
52 8ED0 1C20 5559 : BSBW ADDRCLIST ; Add buffer to receive list
F5 12 1C23 5560 : POPL R2 ; Restore address of next in chain
50 D4 1C26 5561 : BNEQ 110$ ; Br if more in chain
03 08 A6 1C28 5562 120$: CLRL R0 ; Assume failure
FF6F 31 1C2A 5563 : BBS #RCV_STS_V_LAST,- ; Br if NOT end of chain
05 10 1C2C 5564 : RCV_W_STS(R6),130$ ; get rest of message
DB 50 E9 1C32 5565 130$: BRW ASSEM_PKTS ; Else, try for next valid message
EF 11 1C34 5566 : BSBB NEXTMSG ; Get next message
1C37 5567 : BLBC R0,100$ ; Br if none
1C39 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
OB A6 0A A6 91 1C56 5582 10$: CMPB RCV_W_LEN(R6),RCV_W_LEN+1(R6) ; Are we really done?
73 12 1C5B 5583 : BNEQ 90$ ; Br if not, leave now
52 00F4 D4 0F 1C5D 5584 : REMQUE @CDB_Q_RCVPNDR(R4),R2 ; Get next receive
6C 1D 1C62 5585 : BVS 90$ ; Br if none available (yet)
10 A2 D4 1C64 5586 : CLRL CXBSL_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

```

```

      FB 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      ; Compute buffer size
          1C7A 5594      ;
          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          CXBSW_LENGTH(R2)      ;
      0A A6          90 1C82 5599      MOVW  RCV_W_LEN(R6)      ; Store length in CXB <7:0>
      0C A2          1C85 5600          CXBSW_LENGTH(R2)      ;
          2E A0      1C87 5601      ADDW  #XQ_C_ADDRRCV-<XQ_C_HEADER+XQ_C_CRC>,- ; Add in missed count
          0C A2          1C89 5602          CXBSW_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          CXBSW_R_STS(R2)      ;
      01 A6          90 1C90 5605      MOVW  RCV_W_FLAG+1(R6)      ; Save flags byte (high word)
      0B A2          1C93 5606          CXBSW_R_FLAGS(R2)      ;
          1C95 5607      ; Adjust quota and release mapping slot
          1C95 5608      ;
          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
          0108 C4 A0 1CA1 5614      ;
          010A C4 9A 1CA5 5616      30$: ADDW  CDB_W_BSZ(R4)      ; Replenish CDB quota
      51 22 A2      9A 1CA8 5617      40$: MOVZBL CXBSW_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$>,-      ;
          1CB1 5622          <730,80$>,-      ;
          1CB1 .1          <UV2,80$>,-      ;
          1CB1 .2          <UV1,100$>>      ; Copy data on MicroVAX I
          1CCD .3      ;
          1CCD .4      80$: ; All but MicroVAX I
          1CCD .5      ;
          50 01 9A 1CCD .6      MOVZBL S*#SS$_NORMAL,R0      ; Return success!
          05 05 1CD0 .7      90$: RSB
          1CD1 .8      ;
          1CD1 .9      100$: ;*****
          1CD1 .10          ; For MicroVAX I, ONLY.
          1CD1 .11          ;*****
          1CD1 .12          ;
          OE E0 1CD1 .13      BBS  #RCV_STS_V_ERR,-      ; Br if this packet is in error
      F7 14 A2          1CD3 .14          CXBSW_R_STS(R2),80$      ;
          0C A2          B1 1CD6 .15      CMPW  CXBSW_LENGTH(R2),CDB_W_BSZ(R4) ; Check size of received data
          0108 C4 1A 1CDC .16      BGTRU 110$      ; Br if bad size
      50 00C4 C441 D0 1CDE .17      MOVL  CDB_L_RCV_VA(R4)[R1],R0 ; Else, get address of contiguous buffer
          OE A1 1CE4 .18      ADDW3 #XQ_C_HEADER,-      ; Add back in the header
          53 0C A2          1CE6 .19          CXBSW_LENGTH(R2),R3      ;
          34 BB 1CE9 .20      PUSHR #*M<R2,R4,R5>      ; Save registers
          60 53 28 1CEB .21      MOVW3 R3,(R0),CXBST_R_DATA(R2) ; Move the data
          38 A2          1CEE

```

XQDRIVER
V04-001

- VAX/VMS QNA driver
ASSEM_PKTS - Assemble receive packets

D 1

8-JAN-1985 17:49:06 VAX/VMS Macro V04-00 Page 124
5-SEP-1984 00:20:54 [DRIVER.BUGSRC]XQDRIVER.MAR;1 (50)

```
:RNG0001      34  BA  1CF0  .22      POPR  #^M<R2,R4,R5>      ; Restore registers
:RNG0001      D9  11  1CF2  .23      BRB   80$              ; Continue
:RNG0001      05  E0  1CF4  .24
:RNG0001      D3 0246 C4  1CF4  .25 110$:  BBS   #CDB_STS_V_SETUP,-      ; Br if this is a SETUP packet
:RNG0001      1CF6  .26      CDB_B_STSTR4),80$      ; (no error)
:RNG0001      1CFA  .27      SETBIT #RCV_STS_V_ERR,-      ; Else, indicate error
:RNG0001      1CFA  .28      CXBSW_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,-
          1D07 5673      CXB$Q_SIZE(R2)
0168 8F      1D0A      BGTRU   80$                          ; Br if no, ignore message
08 A2      1D0C 5674      MOVL   CDB_L_UCB0(R4),R5            ; Get address of UCB #0
55 0110 C4 D0 1D0E 5675      MOVQ  CXB$G_R_SRC(R2),R0          ; Save source node address
50 3E A2 7D 1D13 5676      MOVW  CXB$T_R_USERDAT+1(R2),-; Copy the request ID
47 A2 B0 1D17 5677      IRP$C_LENGTH+CXB$C_HEADER+1(R2) ; to the Message block
010D C2      1D1A 5678      BSBW  BLD_IRP                    ; Turn the message block into an IRP
0555 30 1D1D 5679      MOVQ  R0,IRP$Q_STATION(R3)           ; Set the return node address
40 A3 50 7D 1D20 5680      MOVL  R4,IRP$L_ASTPRM(R3)         ; Save the CDB address
14 A3 54 D0 1D24 5681      BRB   MOP_CTR_BUILD              ; Build the response message
          0C 11 1D28 5682
          1D2A 5683      : Error - returns packets to receive queue
          1D2A 5684      :
          1D2A 5685      :
10 A2 DD 1D2A 5686      80$: PUSHL  CXB$L_LINK(R2)                ; Save next in chain
F705 30 1D2D 5687      BSBW  ADDRCLIST                    ; 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
1D36 5706      :     R5 = UCB address of UNIT 0
1D36 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      :
1D36 5718      PUSHR #*M<R3,R4,R5,R6,R7,R8> ; Save registers
1D36 5719      MOVL  R4,R6 ; Copy CDB address
53  01F8 8F BB 1D3A 5719      MOVAB  IRP$C_LENGTH+ ; Point to start of message
   56  54  D0 1D3D 5720      :
   010C C3 9E 1D42 5721      : CXB$C_HEADER(R3),R3
   ; ; block
   83  0B  90 1D42 5722      MOVAB  #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  MOPCTR_TAB,R7 ; Get address of MOP counters
   58  87  3C 1D4C 5725 20$: MOVZWL (R7)+,R8 ; Get the offset to the counter desired
   ; ; Br if end of table
   ; 58  19  13 1D4F 5726      BEQL  30$ ; Br if end of table
   ; 67  58  56  C0 1D51 5727      ADDL  R6,R8 ; Compute address of the counter
   ; 07  00  EF 1D54 5728      EXTZV #0,#7,(R7),R2 ; Get the width field without bitmap
   ; ;
63  68  52  28 1D59 5729      MOVC3  R2,(R8),(R3) ; Copy the counter
   ; ; Get the width field again
   ; E8  51  87  9A 1D5D 5730      MOVZBL (R7)+,R1 ; Get the width field again
   ; ; Br if no BITMAP to return
   ; 83  FE A8 B0 1D64 5731      BBC  #7,R1,20$ ; Br if no BITMAP to return
   ; ; Else, store the BITMAP
   ; E2  11 1D68 5732      MOVW  -2(R8),(R3)+ ; Else, store the BITMAP
   ; ; Loop for next counter
   ; 1D6A 5733      BRB  20$ ; Loop for next counter
   ; ;
   ; 01F8 8F BA 1D6A 5734      :
   ; 30$: POPR #*M<R3,R4,R5,R6,R7,R8> ; Restore registers
   ; ;
   ; 226C CF 9E 1D6E 5735      MOVAB  W^DELETE_BLOCK,IRP$C_PID(R3) ; Set the return address
   ; ;
52  00C4 C3 9E 1D74 5738      MOVAB  IRP$C_LENGTH(R3),R2 ; Get address of CXB
   ; ; Make it look like a CXB
   ; OA  A2  1B  90 1D79 5739      MOVAB  #DYN$C_CXB,CXB$B_TYPE(R2) ; Make it look like a CXB
   ; ; Set size of transfer
   ; 005C 8F B0 1D7D 5740      MOVW  #MOP_CTR_SIZE+8+3,- ; Set size of transfer
   ; ;
   ; 1A A2 1D81 5741      :
   ; ; CXB$B_BCNT(R2)
   ; ;
   ; 3A B0 1D83 5742      ASSUME CXB$C_HEADER EQ CXB$T_DATA+XQ_C_HEADER
   ; ; Set offset to start of data
   ; 18 A2 1D85 5743      MOVW  #CXB$T_DATA,- ; Set offset to start of data
   ; ;
   ; 24 A2 53 D0 1D87 5744      :
   ; ; CXB$B_BOFF(R2)
   ; ; Save IRP address in CXB
   ; 40 A3 7D 1D8B 5745      MOVL  R3,CXB$C_IRP(R2) ; Save IRP address in CXB
   ; ; Set STATION in CXB
   ; 28 A2 1D8E 5746      MOVW  IRP$C_STATION(R3),- ; Set STATION in CXB
   ; ; CXB$C_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	MOVL	IRPSL_UCB(R3),R5	; Get the UCB address
52	3A	A2	9E	1D98	MOVAB	CXB\$T-T_DATA(R2),R2	; Set R2 to start of data
	40	A3	7D	1D9C	MOVQ	IRPSQ-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	IRPSC_LENGTH(R3),-	; Insert request on request queue
00E0	D4		1DAA	5757		@CDB_Q_XMTREQ+4(R4)	
	E6D7	31	1DAD	5758	BRW	XMT_ALT_START	; Startup the reply
			1DB0	5759			

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
1DB0 5787 : MOVZBL #DIAG_L_EXTRA,(R0)+ ; Insert number of returned long words
1DB0 5788 : MOVW CDB_L_LASTCSR(R4),(R0)+ ; Insert last CSR contents
1DB0 5789 : MOVZBW CDB_B_DIAG1(R4),(R0)+ ; Return error flags info
1DB0 5790 : MOVW CDB_W_DIAG2(R4),(R0)+ ; Return extra error info
1DB0 5791 : MOVL CDB_G_HWA(R4),(R0)+ ; Return hardware physical address
1DB0 5792 : MOVW CDB_G_HWA+4(R4),(R0)+ ;
1DB0 5793 : RSB ; ...

:RNG0001
-1

80 06 9A 1DB0 5787
80 10 A4 B0 1DB3 .1
80 0119 C4 9B 1DB7 5789
80 011A C4 B0 1DBC 5790
80 0250 C4 D0 1DC1 5791
80 0254 C4 B0 1DC6 5792
05 1DCB 5793

```

1DCC 5795 .SBTTL RESTART_ROUT - PROCESS EXPIRATION (IF 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$_IMEP
1DCC 5811 :
1DCC 5812 : Implicit inputs:
1DCC 5813 :
1DCC 5814 :     IRP$_RBOFF(R5) = UCB address for UNIT
1DCC 5815 :
1DCC 5816 : Outputs:
1DCC 5817 :
1DCC 5818 :     R0-R3 are destroyed.
1DCC 5819 :--
1DCC 5820 :
1DCC 5821 : .ENABL  LSB
RESTART_ROUT::
1DCC 5822 :     PUSHL  R5 ; Process expiration of restart timer
1DCC 5823 :     ASSUME  IRP$_RBOFF GT TQESC_LENGTH ; Save R5
55 52 30 A5 9E 1DCE 5824 :     MOVAB  TQESC_LENGTH(R5),R2 ; Point to IRP portion of TQE
55 0098 C2 D0 1DD2 5825 :     MOVL  IRP$_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$_SIZE EQ 8
1DDA 5831 :     ASSUME  IRP$_TYPE EQ IRP$_SIZE+2
1DDA 5832 :     ASSUME  IRP$_RMOD EQ IRP$_TYPE+1
52 02 C0 1DDA 5833 :     ADDL  #2,R2 ; Skip size field
82 82 0A B0 1DDD 5834 :     MOVW  #DYN$C IRP,(R2)+ ; Set type to IRP
1DE0 5835 :     ASSUME  IRP$_PID EQ IRP$_RMOD+1
82 1E25'CF 9E 1DE0 5836 :     MOVAB  W^RETURN IRP,(R2)+ ; Set return address form IOPOST
1DE5 5837 :     ASSUME  IRP$_AST EQ IRP$_PID+4
1DE5 5838 :     ASSUME  IRP$_ASTPRM EQ IRP$_AST+4
82 7C 1DE5 5839 :     CLRQ  (R2)+ ; Clear AST, ASTPRM
1DE7 5840 :     ASSUME  IRP$_WIND EQ IRP$_ASTPRM+4
1DE7 5841 :     ASSUME  IRP$_UCB EQ IRP$_WIND+4
82 82 D4 1DE7 5842 :     CLRL  (R2)+ ; Clear WIND
82 55 D0 1DE9 5843 :     MOVL  R5,(R2)+ ; Set UCB address
1DEC 5844 :     ASSUME  IRP$_FUNC EQ IRP$_UCB+4
1DEC 5845 :     ASSUME  IRP$_XQ FUNC EQ IRP$_FUNC+1
1DEC 5846 :     ASSUME  IRP$_EFN EQ IRP$_FUNC+2
1DEC 5847 :     ASSUME  IRP$_PRI EQ IRP$_EFN+1
1DEC 5848 :     ASSUME  IRP$_IOSB EQ IRP$_PRI+1
82 0500 8F B0 1DEC 5849 :     MOVW  #<XQ FC_V_RESTART@85>,(R2)+ ; Set function request
82 B4 1DF1 5850 :     CLRW  (R2)+ ; Clear EFN, PRI
82 D4 1DF3 5851 :     CLRL  (R2)+ ; Clear IOSB

```

```

      1DF5 5852      ASSUME IRPSW_CHAN EQ IRPSL_IOSB+4
      1DF5 5853      ASSUME IRPSW_STS EQ IRPSW_CHAN+2
      1DF5 5854      ASSUME IRPSL_SVAPTE EQ IRPSW_STS+2
      82 7C 1DF5 5855      CLRQ (R2)+ ; Clear CHAN, STS, SVAPTE
      1DF7 5856      ASSUME IRPSW_BOFF EQ IRPSL_SVAPTE+4
      1DF7 5857      ASSUME IRPSW_BCNT EQ IRPSW_BOFF+2
      1DF7 5858      ASSUME IRPSW_BCNT EQ IRPSL_BCNT
      1DF7 5859      ASSUME IRPSL_MEDIA EQ IRPSW_BCNT+6
      82 82 7C 1DF7 5860      CLRQ (R2)+ ; Clear BOFF, BCNT
      82 00 3C 1DF9 5861      MOVZWL #<XQ_FC_V_INIT@8>,(R2)+ ; Set MEDIA
      82 82 D4 1DFC 5862      CLRL (R2)+ ; Clear MEDIA+4
      1DFE 5863      ;
      1DFE 5864      ; RESTART the UNIT
      1DFE 5865      ;
      1DFE 5866      ;
      4000 8F AA 1E05 5867      DSBINT UCBSB_FIPL(R5) ; Raise IPL
      68 A5 1E09 5868      BICW #UCBSM_XQ_INTERLOCK,- ; Clear the RESTART interlock
      00 E2 1E0B 5869      BBSS #UCBSV_XQ_INITED,- ; Br if unit already inited
      09 68 A5 1E0D 5870      BSBW UCBSW_DEVSTS(R5),10$ ;
      FOA2 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$: MOVL R3,R5 ; Point R5 to IRP
      07 10 1E1C 5876      BSBB RETURN_IRP ; Return the IRP
      55 8ED0 1E21 5877 30$: ENBINT ; Re-enable interrupts
      05 05 1E24 5878      POPL R5 ; Restore R5
      1E25 5879      RSB ; Return to caller
      1E25 5880      ;
      50 D0 A5 9E 1E25 5881 RETURN_IRP:
      00000000 GF 17 1E25 5882      MOVAB -TQESC_LENGTH(R5),R0 ; Get address of start of structure
      1E29 5883      JMP G^COM$DRVDEALMEM ; Deallocate the IRP

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-7

-1

```

                    .SBTTL TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
1E2F 5885          :++
1E2F 5886          : TQE_TIMER - PROCESS EXPIRATION OF TQE TIMER
1E2F 5887          :
1E2F 5888          : Functional description:
1E2F 5889          :
1E2F 5890          : This routine is entered when the TQE delta time has expired. The action is to
1E2F 5891          : check all timer cells and shut down the controller if any have expired.
1E2F 5892          :
1E2F 5893          : Inputs:
1E2F 5894          :
1E2F 5895          :     R4 = CDB address
1E2F 5896          :     R5 = TQE address
1E2F 5897          :
1E2F 5898          :     IPL = !PLS_TIMER
1E2F 5899          :
1E2F 5900          : Outputs:
1E2F 5901          :
1E2F 5902          :     R0-R3 are destroyed.
1E2F 5903          :     R4,R5 are preserved
1E2F 5904          :--
1E2F 5905          :
1E2F 5906          : .ENABL  LSB
1E2F 5907 TQE_TIMER:: ; Process expiration of TQE timer
1E2F 5908     MOVL  CDB_L_UCB0(R4),R0 ; Get first UCB address
1E34 5909     DSBINT UCBSB_DIPL(R0) ; Sync access to UCB and CDB
1E3B .1          ASSUME CDB_STS_V_INITED EQ 0
1E40 .2          BLBC  CDB_B_STS(R4),STOP_TQE ; Br if NOT inited
1E40 .3
1E40 .12         10$: ;
1E40 .13         ; Check transmit timeout timer
1E40 .14
1E40 .15         TSTB  CDB_B_TIM_XMT(R4) ; Is the xmit timer going?
1E44 .16         BEQL  20$ ; Br if not
1E46 .17         DECB  CDB_B_TIM_XMT(R4) ; Timer expired?
1E4A .18         BEQL  TIMEOUT ; Br if yes
1E4C .19         20$: BBC  #XMSV_STS_BUFFAIL,- ; Br if NO buffer allocation failures
1E4E 5917         CDB_L_DEVDEPEND(R4),30$ ;
1E52 5918         SETIPL CDB_B_FIPL(R4) ; Else, sync access to UCB & CDB
1E56 5919         BSBW  FILCRVLIST ; And try to replenish receive buffers
1E59 5920         30$: ENBINT ; Restore IPL
1E5C 5921
1E5C 5945
1E5C 5946         RSB ; Return to caller
1E5D 5947
1E5D 5948 TIMEOUT: PUSHQ  R4 ; Save R4, R5
1E60 5949         MOVL  R0,R5 ; Copy address of UCB0
1E63 5950         BSBB  DEV_TIMEOUT ; Else, timeout has occurred
1E65 5951         POPQ  R4 ; Restore R4, R5
1E68 5952
1E68 5953 STOP_TQE: ; Stop the TQE
1E68 5954         BICB  #TQESM_REPEAT,TQESB_RQTYPE(R5) ; Stop the timer
1E6C 5955         BICB  #CDB_STS_M_TIMER,- ; Indicate that timer is stopped
1E6E 5956         CDB_B_STS(R4) ;
1E71 5957         BICB  #DPTSM_NOUNLOAD,- ; OKAY to unload the driver now
1E73 5958         DPTSTAB+DPTAB_FLAGS ;
1E78 5959         BRB  30$ ; Leave
1E7A 5960         .DSABL LSB

```

50 0110 C4 D0
28 0246 C4 E9
020A C4 95
06 13
020A C4 97
11 13
0C E1
07 010C C4
F5DA 30
05
55 50 D0
26 10
0B A5 04 8A
08 8A
0246 C4
04 8A
0000000D EF
DF 11

```

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::
1E7A 5984 BICW #UCB$M_TIM!UCB$M_INT,UCB$W_STS(R5) ; Timeout or powerfail
1E7E 5985 ; Disable timer
1E7E 5986 ASSUME UCB$V_XQ_INITED EQ 0
1E7E 5987 BLBC UCB$W_DEVSTS(R5),20$ ; Br if not initied
1E82 5988 ASHL #XQ_SOFT_V_POWER+16,#1,R3 ; Assume powerfail
1E86 5989 BBS #UCB$V_POWER,UCB$W_STS(R5),10$ ; Br if powerfail
1E8B 5990 DEV_TIMEOUT::
1E8B .1 MOVL UCB$L_XQ_CDB(R5),R4 ; Hardware device timeout
1E90 .2 BEQL 20$ ; Get CDB address
1E92 .3 MOVL CDB_L_CSR(R4),R3 ; Br if no CDB
1E97 .4 MOVW #XQ_CSR_M_RESET,CSR(R3) ; Get CSR address
1E9B 5997 ASHL #XQ_SOFT_V_TIMEOUT+16,#1,R3 ; Stop the device
1E9F 5998 SETBIT #XMSV_STS_TIMO,CDB_L_DEVDEPEND(R4) ; Indicate timeout
1EA5 5999 10$: MOVW #XQ_CSR_M_ERR,R3 ; Set error status
1EAA 6000 PUSHL R5 ; Indicate fatal error
1EAC 6001 BSBW SCHED_FORK ; Save UCB address
1EAF 6002 POPL R5 ; Schedule a fork process
1EB2 6003 20$: RSB ; Restore UCB address
1EB3 6004 .DSABL LSB ; Return to caller

```

:RNG0001
:RNG0001
:RNG0001
:RNG0001
-6


```

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:
1EB3 6027 PUSHR #*M<R1,R2,R3,R4,R5> ; Allocate a CDB
1EB3 6028 MOVZWL #CDB_C_LENGTH,R1 ; Save registers
1EB3 6029 JSB G^EXE$ALONONPAGED ; Get size of CDB allocation
1EB3 6030 BLBC R0,90$ ; Try to allocate CDB
1EB3 6031 ; Br if error
1EB3 6032 MOVL UCBSL_CRB(R5),R4 ; Get CRB address
1EB3 6033 MOVL R2,CRBSL_AUXSTRUC(R4) ; Store CDB address
1EB3 6034 MOVL UCBSL_DDB(R5),R4 ; Get DDB address
1EB3 6035 MOVL DDBSL_UCB(R4),R4 ; Get UCBO address
1EB3 6036 MOVL R4,R3 ; Save UCBO address
1EB3 6037 MOVL R2,UCBSL_XQ_CDB(R4) ; Save CDB address in UCBO
1EB3 6038 MOVAB CDB_G_HWA(R2),- ; Store address of NI device's
1EB3 6039 MOVL UCBSL_NI_HWAPTR(R4) ; unique hardware address
1EB3 6040 BNEQ 10$, ; Position to next UCB
1EB3 6041 ; Continue if more UCBS
1EB3 6042 :
1EB3 6043 : Initialize CDB
1EB3 6044 :
1EB3 6045 : The PADDING MODE and the ECHO MODE of the QNA will default to
1EB3 6046 : the enabled (ON) state.
1EB3 6047 :
1EB3 6048 ASSUME NMA$C_STATE_ON EQ 0
1EB3 6049 :
1EB3 6050 PUSHQ R2 ; Save CDB, UCBO address
1EB3 6051 MOVCS #0,(R2),#0,R1,(R2) ; Zero the structure
1EB3 6052 POPQ R2 ; Restore CDB, UCBO address
1EB3 6053 :
1EB3 6054 MOVL R3,CDB_L_UCBO(R2) ; Save address of UCBO
1EB3 6055 :
1EB3 6056 : Init CDB fork block.
1EB3 6057 :
1EB3 6058 ASSUME CDB_L_FQFL EQ 0
1EB3 6059 ASSUME CDB_L_FQBL EQ CDB_L_FQFL+4
1EB3 6060 CLRQ (R2)+ ; Skip link pointers

```

```

51 02F0 8F BB
00000000 GF 16
47 50 E9
54 24 A5 DO
10 A4 52 DO
54 28 A5 DO
54 04 A4 DO
00C8 53 54 DO
C4 52 DO
0250 C2 9E
0090 C4
54 30 A4 DO
EE 12

```

```

:RNG0001
:RNG0001
-1

```

```

00 62 00 2C
62 51
0110 C2 53 DO
82 7C

```

```

083302F0 8F  D0 1EFB 6061  ASSUME CDB_W_SIZE EQ CDB_L_FQBL+4
           82  1EFB 6062  ASSUME CDB_B_TYPE EQ CDB_W_SIZE+2
           1EFB 6063  ASSUME CDB_B_FIPL EQ CDB_B_TYPE+1
           1EFB 6064  MOVL  #<<<IPL$_XQ_FIPL@8>!DYN$C_CDB>@16>!-- ; Set structure type and FIPL
           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 1FOA 6069 90$: POPR  #*M<R1,R2,R3,R4,R5> ; Restore registers
           05  1FOC 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::
:RNG0001 01 0246 C4 E8 1F0D .1 ASSUME CDB_STS_V_INITED EQ 0 ; Shutdown QNA
-2 :RNG0001 05 1F12 6094 BLBS CDB_B_STS(R4),10$ ; Br if QNA inited
:RNG0001 00E8 8F BB 1F13 6095 RSB ; Else, return
1F13 6096 10$: PUSHR #^M<R3,R5,R6,R7> ; Save registers
1F17 6097 :
1F17 6098 : Shutdown QNA and reset controller status
1F17 6099 :
:RNG0001 52 0114 C4 D0 1F17 .1 MOVL CDB_L_CSR(R4),R2 ; Get CSR address
-3 OE A2 02 A8 1F1C 6103 DSBINT UCBSB_DIPL(R5) ; Raise IPL for master clear
F3 8F 8A 1F23 6104 BISW #XQ_CSR_M_RESET,CSR(R2) ; Disable device
0246 C4 1F27 6105 BICB #^C<CDB_STS_M_FORK_PEND!- ; Reset all but needed bits
1F2A :
1F2D 6106 ENBINT CDB_STS_M_TIMER>,CDB_B_STS(R4) ;
1F2D 6107 : Return to fork level
1F30 6108 :
1F30 6109 : Release the receive and transmit buffer map registers
1F30 6110 :
:RNG0001 57 D4 1F30 6111 CLRL R7 ; Init slot number
-1 56 1C A4 9E 1F32 6112 ASSUME CDB_L_RCVMAP+<4*<MAX_C_RCV-1>> EQ CDB_L_XMTMAP
53 24 A5 D0 1F32 .1 MOVAB CDB_L_RCVMAP(R4),R6 ; Get address of mapping slots
1F36 6114 20$: MOVL UCBSL_CRB(R5),R3 ; Get CRB address
1F3A 6115 ASSUME VECSW_MAPREG+2 EQ VECSB_NUMREG
1F3A 6116 ASSUME VECSB_NUMREG+1 EQ VECSB_DATAPATH
34 A3 86 D0 1F3A 6117 MOVL (R6)+,CRBSL_INTD+VECSW_MAPREG(R3) ; Set mapping information
1B 19 1F3E 6118 BLSS 30$ ; Br if none allocated
37 A3 95 1F40 6119 TSTB CRBSL_INTD+VECSB_DATAPATH(R3) ; Is there a datapath?
OC 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
FC A6 01 CE 1F51 6123 25$: RELMPR ; Release the map register
1F57 6124 MNEGL #1,-4(R6) ; Reset mapping info
1F5B 6125 ASSUME MAX_C_RCV EQ 8
:RNG0001 D2 57 0A F2 1F5B 6126 30$: CLRBIT R7,CDB_B_RCVMAP(R4) ; Clear mapping slot flag
-1 1F60 .1 AOBLS #<MAX_C_RCV-1>+<MAX_C_XMT-1>,R7,20$ ; Loop if more map registers
1F64 6128 :
1F64 6129 : Release the PCBB map registers

```

```

53 24 A5 D0 1F64 6130 ;
1F64 6131 ;
1F68 6132 ;
1F68 6133 ;
011C C4 D0 1F68 6134 ;
34 A3 1F6C 6135 ;
0A 13 1F6E 6136 ;
011C C4 D4 1F70 6137 ;
1F74 6138 ;
1F7A 6139 50$: ;
1F7A 6140 ;
1F7A 6141 ;
56 00DC C4 9E 1F7A 6142 ;
57 05 3C 1F7F 6143 ;
53 00 B6 0F 1F82 6144 60$: ;
79 1D 1F86 6145 ;
0A 0A A3 91 1F88 6146 ;
0A 13 1F8C 6147 ;
1B 0A A3 91 1F8E 6148 ;
09 13 1F92 6149 ;
1F94 6150 ;
1F98 6151 ;
1F98 6152 ; IRP
1F98 6153 ;
FB9E 30 1F98 6154 70$: ;
E5 11 1F9B 6155 ;
1F9D 6156 ;
1F9D 6157 ; CXB
1F9D 6158 ;
1F9D 6159 80$: ;
1F9D 6160 ;
1F9D 6161 ;
1F9D 6162 ;
1F9D 6163 ;
1F9D 6164 ;
1F9D 6165 ;
1F9D 6166 ;
1F9D 6167 ;
1FB0 6168 ;
1FB0 6169 ;
1FB4 6170 ;
1FB4 6171 ;
1FB4 6172 ; CXB - XMIT request
1FB4 6173 ;
1FB4 6174 100$: ;
16 24 A3 E8 1FB4 6175 ;
53 24 A3 D0 1FB8 6176 ;
50 0094 C3 D0 1FBC 6177 ;
06 13 1FC1 6178 ;
0000000 GF 16 1FC3 6179 ;
FB6D 30 1FC9 6180 110$: ;
B4 11 1FCC 6181 ;
1FCE 6182 ;
1FCE 6183 120$: ;
1FCE 6184 ;
1FCE 6185 ;
51 18 A3 3C 1FCE 6186 ;

MOVLC UCBSL_CRB(R5),R3 ; Get CRB address
ASSUME VEC$W_MAPREG+2 EQ VEC$B_NUMREG
ASSUME VEC$B_NUMREG+1 EQ VEC$B_DATAPATH
MOVL CDB_L_RINGMAP(R4),- ; Setup map info in CRB
CRB$_INTD+VEC$W_MAPREG(R3) ;
BEQL 50$ ; Br if none
CLRL CDB_L_RINGMAP(R4) ; No more mapping info
RELMR ; Release the map register
; Deallocate all receive buffers and complete all I/O request packets
MOVAB CDB_Q_QUEUES(R4),R6 ; Get address of first queue listhead
MOVZWL #CDB_C_QUEUES,R7 ; Get number of queues
REMQUE @ (R6),R3 ; Get next IRP/BUFFER
BVS 150$ ; Br if none
CMPB IRP$_TYPE(R3),S^#DYN$_IRP ; Is this an IRP?
BEQL 70$ ; Br if yes
CMPB IRP$_TYPE(R3),S^#DYN$_CXB ; Is this a CXB?
BEQL 80$ ; Br if yes
BUG_CHECK NOBUF_PCKT,FATAL ; Else, fatal error

BSBW ABORT_PKT ; Abort the IRP
BRB 60$ ; Try for more

$DISPATCH CXB$_XQ_FUNC(R3),TYPE=B,-
<.. ;function action
<XQ_FC_V_XMIT 100$>,- ; XMIT request
<XQ_FC_V_RECV 140$>,- ; RECV request
<XQ_FC_V_INIT 100$>,- ; INIT request
<XQ_FC_V_STOP 100$>,- ; STOP request
<XQ_FC_V_CHMODE 100$>,- ; Change mode request
>

BUG_CHECK NOBUF_PCKT,FATAL ; Fatal error - not a valid IRP

ASSUME CXB$_T_IRP EQ CXB$_T_UCB
BLBS CXB$_T_IRP(R3),120$ ; Br if not IRP address => FFI user
MOVL CXB$_T_IRP(R3),R3 ; Else, get IRP address
MOVL IRP$_XQ_SETUP(R3),R0 ; Get SETUP mode buffer
BEQL 110$ ; Br if not present
JSB G^COM$DRVDEALMEM ; Else, deallocate buffer
BSBW ABORT_PKT ; Abort the IRP
BRB 60$ ; Try for more

; This is an FFI user, return CXB buffer
MOVZWL CXB$_BOFF(R3),R1 ; Get offset to start of data

```

```

51 51 53 C0 1FD2 6187 ADDL R3,R1 ; Compute start of Ethernet header
    OC A1 3C 1FD5 6188 MOVZWL XBUF_W_TYPE(R1),R1 ; Get protocol type
    OABF 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 FUSHL R4 ; Save R4
54 0192 C5 D0 1FE1 6192 MOVL UCBSL_XQ_FFI(R5),R4 ; Get FFI block address
    OB 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 @FFISL_XMIT_DONE(R4) ; Complete the XMIT
    54 BEO0 1FEE 6196 POPL R4 ; Restore R4
    BF 11 1FF1 6197 BRB 60$ ; Try for more
    1FF3 6198 130$: ;
    1FF3 6199 ; No more FFI for CXB to complete
    1FF3 6200 ;
    54 BEO0 1FF3 6201 POPL R4 ; Restore CDB address
    1FF6 6202 ;
    1FF6 6203 ; CXB - RECV request
    1FF6 6204 ;
    1FF6 6205 ;
    1FF6 6206 ;
    50 53 D0 1FF6 6207 140$: 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$: ADDL #8,R6 ; Skip to next queue listhead
    FB 57 F5 2004 6214 SOBGTR R7,1060$ ; 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
    OB 13 200C 6219 BEQL 190$ ; Br if none, yet
    55 30 A5 D0 200E 6220 170$: MOVL UCBSL_LINK(R5),R5 ; Get next unit's address
    05 13 2012 6221 BEQL 190$ ; Br if none
    0026 30 2014 6222 BSBW SHUTDOWN ; Shutdown the UNIT
    F5 11 2017 6223 BRB 170$ ; Check if more UNIT's
    00E8 8F BA 2019 6224 190$: POPR #*M<R3,R5,R6,R7> ; Restore registers
    05 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::
19 64 A5 E1 201E 6254      BBC      #UCBSV_ONLINE,-          ; Shut down protocol type
2020 6255      UCBSW_STS(R5),10$          ; Br if not online
2023 6256      ASSUME UCBSV_XQ_INITED EQ 0
15 68 A5 E9 2023 6257      BLBC UCBSW_DEVSTS(R5),10$      ; Br if not inited
03 E1 2027 6258      BBC      #UCBSV_XQ_SHARE,-          ; Br if not a shared UCB
11 68 A5 2029 6259      UCBSW_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 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$: RSB          ; Return to caller
203D 6273      :
203D 6274      SHUTDOWN::          ; Shut down unit
04 64 A5 E1 203D 6275      BBC      #UCBSV_ONLINE,-          ;
203F 6276      UCBSW_STS(R5),5$          ; If BC not online
2042 6277      :
2042 6278      ASSUME UCBSV_XQ_INITED EQ 0
01 68 A5 E8 2042 6279      BLBS UCBSW_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 :
19 64 05 E1 2047 6289 10$: BBC #UCBSV_POWER,- ; Skip notification if power failure
A5 2049 6290 UCBSW_STS(R5),20$ ; did not occur
54 0192 54 DD 204C 6291 PUSHL R4 ; Save CDB address
C5 DO 204E 6292 MOVL UCBSL_XQ_FFI(R5),R4 ; Retrieve FFI address
OD 13 2053 6293 BEQL 15$ ; Nothing to do if there isn't one
52 1C A4 DO 2055 6294 MOVL FFISL_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$: POPL R4 ; Restore CDB address
2065 6299 :
2065 6300 :
2065 6301 : Start a 3-second timer to restart any UNIT needing automatic restart.
2065 6302 : This restart timer only runs if the device was halted due to a fatal error.
2065 6303 :
2065 6304 : Note, that the UCB multicast address list is purged, which will nullify
2065 6305 : any restart operation that may be performed by the QNA driver itself (only
2065 6306 : if the user has specified any multicast addresses).
2065 6307 :
00C8 8F BB 2065 6309 20$: PUSHR #^M<R3,R6,R7> ; Save registers
OF E1 2069 6310 BBC #UCBSV_XQ_RESTART,- ; Br if this UNIT does not need
43 68 A5 206B 6311 UCBSW_DEVSTS(R5),22$ ; automatic restart
206E 6312 ASSUME IRP$C_LENGTH GE TQESC_LENGTH
51 F4 8F 9A 206E 6313 MOVZBL #IRP$C_LENGTH+TQESC_LENGTH,R1 ; Get size of IRP/TQE
00000000 GF 16 2072 6314 JSB G^EXESALONONPAGED ; Try to allocate a IRP/TQE
36 50 E9 2078 6315 BLBC R0,22$ ; Br if failure - too bad
4000 8F AB 207B 6316 ASSUME IRP$Q_STATION GT TQESC_LENGTH
68 A5 207B 6317 BISW #UCBSM_XQ_INTERLOCK,- ; Interlock the RESTART bit
207F 6318 UCBSW_DEVSTS(R5)
2081 6319 :
2081 6320 ASSUME TQESB_TYPE EQ TQESW_SIZE+2
2081 6321 ASSUME TQESB_RQTYPE EQ TQESB_TYPE+1
000F00F4 8F DO 2081 6322 MOVL #<<DYNSC_TQE@16>!<IRP$C_LENGTH+TQESC_LENGTH>>,- ; Set STRUCTURE
08 A2 2087 6323 TQESW_SIZE(R2) ; TYPE and SIZE
00C8 C2 55 DO 2089 6324 MOVL R5,TQESC_LENGTH+IRP$RBOFF(R2) ; Save UCB address in IRP
55 52 DO 208E 6325 PUSHQ R4 ; Save R4, R5
7D 2091 6326 MOVL R2,R5 ; Copy TQE address
00 01C9C380 8F 7D 2094 6327 MOVQ #RESTART_DELTA,- ; Set the delta time
2095 :
209A :
53 20 A5 209E 6328 TQESQ_DELTA(R5)
FD28 CF 9E 20A0 6329 MOVAB W^RESTART_ROUT,R3 ; Get address of RESTART routine
01 90 20A5 6330 MOVB #TQESC_SSSNGL,- ; Set the request type
2C A5 20A7 6331 TQESL_RQPID(R5)
F275 30 20A9 6332 BSBW FORK_TIMER ; FORK to startup the timer
05 11 20AC 6333 POPQ R4 ; Restore R4, R5
20AF 6334 BRB 23$ ; Contine
20B1 6335 :
20B1 6336 22$: CLRBIT #UCBSV_XQ_RESTART,- ; Restart is not possible!
20B1 6337 UCBSW_DEVSTS(R5)
2328 8F B0 20B6 6338 23$: MOVW #INIT_C_QUOTA,UCBSW_XQ_HBQ(R5) ; Reset hardware buffer quota

```

```

00D6 C5      20BA
64 A5 23 AA 20BD 6339      BICW      #UCBSM_INT!UCBSM POWER!-
                                UCBSM_TIM,UCBSW_STS(R5) ; Reset device status
68 A5 11 AA 20C1 6340      BICW      #UCBSM_XQ_INITED!- ; No longer initied
                                UCBSM_XQ_RUN,UCBSW_DEVSTS(R5) ; or running
010C C4 10 E1 20C5 6342      CLRBIT   #XMSV_STS_ACTIVE,UCBSL_DEVDEPEND(R5) ; Clear active bit
                                05 20CA 6343      BBC      #XMSV_ERR_FATAL,CDB_L_DEVDEPEND(R4),25$ ; Br if not FATAL
                                20CF 6344      SETBIT  #XMSV_ERR_FATAL,UCBSL_DEVDEPEND(R5) ; Else, indicate FATAL
                                20D0 6345      ;
                                20D5 6346      ; Reset UCB multicast address list
                                20D5 6347      ;
                                20D5 6348      ;
0210 C4 55 D1 20D5 6349 25$: CMPL      R5,CDB_L_PRMUSER(R4) ; Is this unit the PROMISCUOUS user?
                                OC 12 20DA 6350      BNEQ      27$ ; Br if not
0210 C4 D4 20DC 6351      CLRL     CDB_L_PRMUSER(R4) ; Else, clear the PROMISCUOUS user addr
                                01 90 20E0 6352      MOVVB    #NMASC_STATE_OFF,- ; Don't forget about the CDB
0247 C4 01 20E2 6353      ; parameter
                                0154 30 20E5 6354      BSBW    BLD_STOP_IRP ; Build an IRP to RESET hardware mode
                                OF E0 20E8 6355 27$: BBS      #UCBSV_XQ_RESTART,- ; Br if this UNIT is restarting
                                16 68 A5 20EA 6356      UCBSW_DEVSTS(R5),28$ ; don't clear multicast list
                                00E9 C5 94 20ED 6357      PUSHQ   R4 ; Save R4, R5
00EC C5 00 2C 20F0 6358      CLRB    UCBSB_XQ_MULTI(R5) ; No more multicast addresses
                                00 20F4 6359      MOVCS   #0,UCBSG_XQ_MULTI(R5),#0,- ; Zero the structure
                                0048 8F 20F9 ;
                                00EC C5 20FA 6360      #6*MAX_C_MLT,UCBSG_XQ_MULTI(R5) ;
                                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   UCBSL_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  ACBSL_KAST+10(R0),R6 ; Force channel match
                                52 24 A0 3C 2116 6371      MOVZWL  ACBSL_KAST+12(R0),R2 ; Get process index
                                00000000 GF D0 211A 6372      MOVL    G^SCH$GL_PCBVEC,R4 ; Get PCB address vector address
                                54 54 2120 ;
                                54 6442 D0 2121 6373      MOVL    (R4)[R2],R4 ; Get PCB address
                                00000000 GF 16 2125 6374      JSB     G^COM$FLUSHATTNS ; Flush AST
                                DB 11 212B 6375      BRB     30$ ;
                                54 8ED0 212D 6376 40$: POPL    R4 ; Restore CDB address
                                2130 6377      ;
                                2130 6378      ; Complete all RCV IRPs for this unit
                                2130 6379      ;
                                53 00A8 D5 OF 2130 6380      ASSUME  UCBSQ_XQ_QUEUES-1 EQ 3 ; One queue for shared users
                                05 1D 2135 6381 45$: REMQUE  @UCBSQ_XQ_RCVREQ(R5),R3 ; Get IRP
                                F9FF 30 2137 6382      BVS     50$ ; Br if none
                                F4 11 213A 6383      BSBW   ABORT_PKT ; Abort the I/O request
                                213C 6384      BRB     45$ ; Get next IRP
                                213C 6385      ;
                                213C 6386      ; Complete all XMIT CXBs for this unit
                                213C 6387      ;
53 00B0 D5 OF 213C 6388 50$: REMQUE  @UCBSQ_XQ_XMTREQ(R5),R3 ; Get CXB
                                2C 1D 2141 6389      BVS     55$ ; Br if none
                                2143 6390      ASSUME  CXBSL_T_IRP EQ CXBSL_T_UCB

```



```

12 24 A3 E9 2143 6391 BLBC CXBSL_T_IRP(R3),53$ : Br if IRP address
54 0192 C5 DD 2147 6392 PUSHL R4 : Save CDB address
50 50 2C DO 2149 6393 MOVL UCBSL_XQ_FFI(R3),R4 : Get FFI block address
14 B4 3C 214E 6394 MOVZWL #SS$ ABORT,R0 : Set status return
54 8ED0 16 2151 6395 JSB @FFISL_XMIT_DONE(R4) : Complete CXB
E3 11 2154 6396 POPL R4 : Restore CDB address
2157 6397 BRB 50$ : Get next CXB
2159 6398
53 24 A3 DO 2159 6399 53$: MOVL CXBSL_T_IRP(R3),R3 : Get IRP address
50 0094 C3 DO 215D 6400 MOVL IRPSL_XQ_SETUP(R3),R0 : Get SETUP mode buffer
06 13 2162 6401 BEQL 54$ : Br if none
00000000 GF 16 2164 6402 JSB G^COMSDRVDEALMEM : 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 OF 216F 6408 55$: REMQUE @UCBSQ_XQ_RCVMSG(R5),R2 : Get message buffer
24 1D 2174 6409 BVS 70$ : Br if none
42 A5 AO 2176 6410 ADDW UCBSW_DEVBUSIZ(R5),- : Restore quota
00CC C5 2179 6411 UCBSW_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 #CXBSC_HEADER+- : Calculate size of "normal"
217D 6419 CXBSC_TRAILER,- : receive buffer
217D 6420 CDB_W_BSZ(R4),R0 :
50 004C 8F 2180 6421 CMPW R0,CXBSW_SIZE(R2) : Can buffer be returned?
0108 C4 50 B1 2184 6421 BNEQ 60$ : Br if not, delete buffer instead
08 A2 50 12 2188 6422 BSBW ADDRCLIST : Try to add to receiver list
F2AB 30 218A 6423 BRB 50$ : Loop for more
AD 11 218D 6424
218F 6425
50 52 DO 218F 6426 60$: MOVL R2,R0 : Copy buffer address for deallocation
00000000 GF 16 2192 6427 JSB G^COMSDRVDEALMEM : Deallocate the buffer
A2 11 2198 6428 BRB 50$ : Loop for more
219A 6429 :
219A 6430 : Cleanup all SHR structures if fatal error
219A 6431 :
70$: BBC #UCBSV_XQ_SHARE,- : Br if not a SHARED UCB
30 68 A5 E1 219A 6432 UCBSW_DEVSTS(R5),100$ :
56 00C4 C5 DO 219F 6434 MOVL UCBSL_XQ_DEFUSR(R5),R6 : Get default SHR structure address
08 13 21A4 6435 BEQL 90$ : Br if none
00E2 30 21A6 6436 BSBW CLEANUP_SHR : Else, cleanup the structure
10 E1 21A9 6437 BBC #XMSV_ERR_FATAL,- : Br if not a fatal error
03 44 A5 21AB 6438 UCBSL_DEVDEPEND(R5),90$ :
00FC 30 21AE 6439 BSBW DELETE_SHR : And delete the structure
56 0098 C5 DO 21B1 6440 90$: MOVL UCBSQ_XQ_SHARE(R5),R6 : Get address of next LIMITED user
0098 C5 56 D1 21B6 6441 CMPL R6,UCBSQ_XQ_SHARE(R5) : End of list?
45 13 21BB 6442 BEQL 120$ : Br if yes, don't restore quota (yet)
00CB 30 21BD 6443 BSBW CLEANUP_SHR : Cleanup the I/O
10 E1 21C0 6444 BBC #XMSV_ERR_FATAL,- : Br if not a fatal error
EC 44 A5 21C2 6445 UCBSL_DEVDEPEND(R5),90$ :
OF E0 21C5 6446 BBS #UCBSV_XQ_RESTART,- : Br if this UNIT is re-starting

```

```

E7 68 A5      21C7 6447      UCBSW_DEVSTS(R5),90$
      00E0 30 21CA 6448      BSBW      : Else, delete the structure
      E2 11 21CD 6449      BRB      : Look for more
      21CF 6450      :
      21CF 6451      : Restore quota
      21CF 6452      :
      5A 68 A5  OF E0 21CF 6453 100$: BBS      #UCBSV XQ RESTART, -      : Br if this UNIT is re-starting
50 00B8 C5 3C 21D1 6454      UCBSW_DEVSTS(R5),140$
00000000 GF DO 21D4 6455      MOVZWL    UCBSL_XQ_PID(R5),R0      : Get PID of last starter
      51 21D9 6456      MOVL     G^SCH$GL_PCBVEC,R1      : Address PCB vector
      50 6140 DO 21E0 6457      MOVL     (R1)[R0],R0      : Get PCB of owner
      60 A0 D1 21E4 6458      CMPL    PCB$S_PID(R0),-      :
      00B8 C5 21E7 6459      UCBSL_XQ_PID(R5)      : Still there?
      16 12 21EA 6460      BNEQ    120$      : If NEQ no
50 0080 C0 DO 21EC 6461      MOVL    PCB$S_JIB(R0),R0      : Get JIB address
51 00CC C5 3C 21F1 6462      MOVZWL  UCBSW_XQ_QUOTA(R5),R1      : Convert to longword
      20 A0 51 C0 21F6 6463      ADDL    R1,JIB$S_BYTCNT(R0)      : Return byte count quota
      24 A0 51 C0 21FA 6464      ADDL    R1,JIB$S_BYTLM(R0)      : ..and byte limit quota
      00CC C5 B4 21FE 6465      CLRW   UCBSW_XQ_QUOTA(R5)      : Prevent this from being
      2202 6466      : returned again
      2202 6467      :
      2202 6468      : Delete the STARTUP IRP for point-to-point mode
      2202 6469      :
50 0196 C5 DO 2202 6470 120$: MOVL    UCDSL_XQ_STIRP(R5),R0      :% Get the startup IRP address
      0A 13 2207 6471      BEQL    130$      :% Br if none
      0196 C5 D4 2209 6472      CLRL   UCBSL_XQ_STIRP(R5)      :% All done
00000000 GF 16 220D 6473      JSB    G^COM$DRVDEALMEM      :% Deallocate the IRP
      2213 6474      :
      2213 6475      : If there is an FFI block and the SHUT_DONE routine is set, then
      2213 6476      : notify the FFI user that shutdown is now complete.
      2213 6477      :
50 0192 C5 DO 2213 6478 130$: MOVL    UCBSL_XQ_FFI(R5),R0      : Get FFI block address
      14 13 2218 6479      BEQL    140$      : Br if none
      0192 C5 D4 221A 6480      CLRL   UCBSL_XQ_FFI(R5)      : Cleanup FFI interface
51 20 A0 DO 221E 6481      MOVL   FFI$S_SHUT_DONE(R0),R1      : Get address of routine
      0A 13 2222 6482      BEQL    140$      : Br if none
      54 54 DD 2224 6483      PUSHL  R4      : Save CDB address
      54 50 DO 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      :
      020B 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 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 promiscuous.
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$ALONONPAGED to allocate the IRP because the other
223C 6517 : routines reset the IPL to ASTDEL.
223C 6518 :
223C 6519 BBS #XMSV_ERR_FATAL,- ; Br if fatal error,
223E 6520 CDB_L_DEVDEPEND(R4),10$ ; ignore reset of mode
2242 6521 MOVZWL #IRP$C_LENGTH,R1 ; Set length of IRP
2247 6522 JSB G*EXE$ALONONPAGED ; Try to allocate an IRP
224D 6523 BLBS R0,20$ ; Okay if buffer allocated
2250 6524 10$: RSB ; Else, too bad if we can't do it
2251 6525
2251 6526 20$: MOVW R1,IRP$W_SIZE(R2) ; Fill in the size field
2255 6527 BSBB BLD_IRP ; Build a template IRP
2257 6528 MOVAB B*DELETE_BLOCK,IRP$L_PID(R3) ; Store return address from IOPOST
225C 6529 BSBW SETUP_MODE ; Allocate setup mode buffer
225F 6530 PLBC R0,70$ ; Leave on error
2262 6531 MOVB #XQ_FC_V_STOP,- ; Set function request,
2264 6532 CXB$B_XQ_FUNC(R2) ; looks like a STOP
2266 6533 RSB ; Return to queue request to DEQNA
2267 6534
2267 6535 70$: MOVL R3,R0 ; Copy IRP address
226A 6536 BRB 90$ ; Deallocate IRP
226C 6537
226C 6538 DELETE_BLOCK: ; Deallocate a data structure
226C 6539 MOVL R5,R0 ; Get address of structure
226F 6540 90$: JMP G*COM$DRVDEALMEM ; Deallocate the structure
2275 6541
2275 6542 .DSABL LSB

```

```

10 E0
OE 010C C4
51 00C4 8F 3C
00000000'GF 16
01 50 E8
05
08 A2 51 B0
1E 10
OC A3 6C'AF 9E
F105 30
05 50 E9
03 90
20 A2 05
50 53 D0
03 11
50 55 D0
00000000'GF 17

```

```
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 :--
2275 .2
```

:RNG0001

```

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:
53  82  7E 2275 6607      MOVAQ   (R2)+,R3          ; Build an IRP
2278 6608      ASSUME   IRPSW_SIZE EQ 8          ; Save IRP address, skip to size field
2278 6609      ASSUME   IRPSB_TYPE EQ IRPSW_SIZE+2
2278 6610      ASSUME   IRPSB_RMOD EQ IRPSB_TYPE+1
82  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   IRPSL_PID EQ IRPSB_RMOD+1
227D 6614      ASSUME   IRPSL_AST EQ IRPSL_PID+4
82  7C  227D 6615      CLRQ    (R2)+          ; Clear PID, AST
227F 6616      ASSUME   IRPSL_ASTPRM EQ IRPSL_AST+4
227F 6617      ASSUME   IRPSL_WIND EQ IRPSL_ASTPRM+4
82  7C  227F 6618      CLRQ    (R2)+          ; Clear ASTPRM, WIND
2281 6619      ASSUME   IRPSL_UCB EQ IRPSL_WIND+4
82  55  D0 2281 6620      MOVL    R5,(R2)+      ; Store UCB address
2284 6621      ASSUME   IRPSW_FUNC EQ IRPSL_UCB+4
2284 6622      ASSUME   IRPSB_EFN EQ IRPSW_FUNC+2
2284 6623      ASSUME   IRPSB_PRI EQ IRPSB_EFN+1
2284 6624      ASSUME   IRPSL_IOSB EQ IRPSB_PRI+1
82  7C  2284 6625      CLRQ    (R2)+          ; Clear FUNC, EFN, PRI, IOSB
2286 6626      ASSUME   IRPSW_CHAN EQ IRPSL_IOSB+4
2286 6627      ASSUME   IRPSW_STS EQ IRPSW_CHAN+2
2286 6628      ASSUME   IRPSL_SVAPTE EQ IRPSW_STS+2
82  7C  2286 6629      CLRQ    (R2)+          ; Clear CHAN, STS, SVAPTE
2288 6630      ASSUME   IRPSW_BOFF EQ IRPSL_SVAPTE+4
2288 6631      ASSUME   IRPSW_BCNT EQ IRPSW_BOFF+2
2288 6632      ASSUME   IRPSL_BCNT EQ IRPSW_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::
53 DD 228B 6659      PUSH  R3          ; Cleanup all I/O on SHR structure
228D 6660      ; Save R3
228D 6661      :
228D 6662      :
228D 6663      :
228D 6664      :     ASSUME SHR_C_QUEUES EQ 2
228D 6665      :
228D 6666      : Complete all IRPs for this structure
228D 6667      :
53 20 B6 OF 228D 6668 10$: REMQUE @SHR_Q_RCVREQ(R6),R3 ; Get IRP
   05 1D 2291 6669      BVS  20$ ; 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 OF 2298 6705 20$: REMQUE @SHR_Q_RCVMMSG(R6),R2 ; Get message buffer
   0B 1D 229C 6706      BVS  30$ ; Br if none
   42 A5 A0 229E 6707      ADDW UCBSW_DEVBUFSIZ(R5),- ; Restore quota
   00CC C5 22A1 6708      UCBSW_XQ_QUOTA(R5) ;
   F18E 30 22A4 6709      BSBW ADDRRCVLIST ; Try to add to receiver list
   EF  11 22A7 6710      BRB  20$ ; Loop
   53 8ED0 30$: 22A9 6711      POPL  R3 ; Restore R3
   05  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 ; RO-R1 are destroyed.
22AD 6729 ; All other registers are preserved.
22AD 6730 ;--
22AD 6731 DELETE_SHR::
00C4 5C A5 B7 22AD 6732 DECW UCBSW REFC(R5) ; Delete SHR data structure
C5 56 D1 22B0 6733 CMPL R6,UCBSL_XQ_DEFUSR(R5) ; One less user of the unit
1C 13 22B5 6734 BEQL 30$ ; Is this the default user?
51 0098 C5 9E 22B7 6735 MOVAB UCBSQ_XQ_SHARE(R5),R1 ; Br if yes
50 61 D0 22B8 6736 MOVL (R1),R0 ; Get address of SHARE queue
51 50 D1 22BF 6737 10$: CMPL R0,R1 ; Get address of next in queue
4D 13 22C2 6738 BEQL 90$ ; Back to front of list?
50 56 D1 22C4 6739 CMPL R6,R0 ; Br if none found
05 13 22C7 6740 BEQL 20$ ; Is this the one?
50 60 D0 22C9 6741 MOVL (R0),R0 ; Br if yes
F1 11 22CC 6742 BRB 10$ ; Else, get next in queue
50 60 OF 22CE 6743 20$: REMQUE (R0),R0 ; And try for match
04 11 22D1 6744 BRB 40$ ; Remove structure from list
00C4 C5 D4 22D3 6745 30$: CLRL UCBSL_XQ_DEFUSR(R5) ; And delete the structure
50 0C A6 3C 22D7 6746 40$: MOVZWL SHR_L_PID(R6),R0 ; No more default user
00000000'GF D0 22DB 6747 MOVL G^SCH$GL_PCBVEC,R1 ; Get PID SHR structure
51 22E1 ; Address PCB vector
50 6140 D0 22E2 6748 MOVL (R1)[R0],R0 ; Get PCB of owner
60 A0 D1 22E6 6749 CMPL PCBSL_PID(R0),- ;
OC A6 22E9 6750 SHR_L_PID(R6) ; Still there?
1B 12 22EB 6751 BNEQ 60$ ; If NEQ no
5C 0080 C0 D0 22ED 6752 MOVL PCBSL_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,UCBSW_XQ_QUOTA(R5) ; Decrease the current quota
0190 C5 51 A2 2303 6757 SUBW R1,UCBSW_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 NOBUFCKT,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::
2315 6792     PUSH  #^M<R3,R4,R6,R7>          ; Cancel I/O
2315 6793     BBC   #UCB$V_XQ_SHARE, -      ; Save registers
2315 6794     UCB$W_DEVSTS(R5),2$          ; Br if not a shared UCB
2315 6795     ; perform regular $CANCEL
2315 6796 :
2315 6797 : Try to find SHR data structure
2315 6798 :
2315 6799     BSBW  FIND_SHR                ; Check PID and CHAN
2315 6800     BNEC  2$                     ; Br if NO MATCH, maybe last $DASSGN
2315 6801 :
2315 6802 : Match found - clear inited bit and clean up all I/O on SHR data
2315 6803 : structure.
2315 6804 :
2315 6805 : We will Delete the SHR structure if this is a $DASSGN function
2315 6806 : request. We will get this function when called from SYSSDASSGN
2315 6807 : system service and so we will have to delete the SHR structure
2315 6808 : and decrement the reference count. Note that the reference count
2315 6809 : can never reach zero. Therefore, SYSSDASSGN will decrement the
2315 6810 : reference count on exit and we will be called again. This time
2315 6811 : there will be no match on the PID/CHAN and so the UCB will be
2315 6812 : cleaned up and deleted.
2315 6813 :
2315 6814     MOVL  R1,R6                    ; Copy SHR address
2315 6815     MOVL  UCB$L_XQ_CDB(R5),R4      ; Get CDB address
2315 6816     BSBW  CLEANUP_SHR             ; Cleanup the SHR data structure
2315 6817 :
2315 6818     ASSUME CAN$C_DASSGN EQ 1
2315 6819     DECL  R8                       ; Deassign request?
2315 6820     BNEQ  10$                     ; Br if no - all done
2315 6821     BSBW  DELETE_SHR             ; Else, delete the SHR data structure
2315 6822     ; And NOW perform like a NON-SHARED
2315 6823     ; unit.

```

00D8 8F BB
03 E1
17 68 A5

0136 30
12 12

:RNG0001
-2

54 56 51 D0
00C8 C5 D0
FF5D 30
58 D7
33 12
FF78 30

:RNG0001
:RNG0001
:RNG0001
:RNG0001
-5

```

2335 6824 :
2335 6825 : Non-shared unit - perform $CANCEL function.
2335 6826 :
2335 6827 2$: TSTW UCB$W_REFC(R5) : Last reference?
2338 6828 BNEQ 20$ : Br if no - do selective cancel
54 00C8 C5 D0 233A .1 3$: MOVL UCB$L_XQ_CDB(R5),R4 : Get CDB address
FCFB 30 233F .2 BSBW SHUTDOWN : Shutdown entire unit
.3 ASSUME CDB_STS_V_INITED EQ 0
2342 .4 BLBS CDB_B_STSTR(R7),10$ : Br if QNA is still inited
2342 .5 MNEGL #1,CDB_G_PHA(R4) : Reset physical address
0A 0246 C4 E8 2347 6834 MNEGW #1,CDB_G_PHA+4(R4) :
024A C4 01 CE 234C 6835 :
024E C4 01 AE 2351 6836 :
2351 6837 : When this is the last reference to the unit, reset the CPID of the UCB.
2351 6838 :
2351 6839 5$: BISW S^#UCB$M_ONLINE,- : Set the UNIT to ONLINE
64 A5 A8 2353 6840 UCB$W_STS(R5) :
00BC C5 D5 2355 6841 TSTL UCB$L_XQ_CPID(R5) : Did we save the Creator PID?
OA 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
OS 2369 6846 RSB :
236A 6847 : Abort all associated receive packets on UCB queue
236A 6848 :
236A 6849 :
54 04 AE D0 236A .1 20$: MOVL 4(SP),R4 : Restore PCB address to R4
04 E1 236E .2 BBC #UCB$V_ONLINE,- : Br if not online
F2 64 A5 2370 6851 UCB$W_STS(R5),10$ :
2373 6852 :
2373 6853 ASSUME UCB$V_XQ_INITED EQ 0
56 EE 68 A5 E9 2373 6854 BLBC UCB$W_DEVSTS(R5),10$ : Br if not inited
00A8 C5 9E 2377 6855 MOVAB UCB$Q_XQ_RCVREQ(R5),R6 : Get address of receive queue
29 10 237C 6856 BSBB CHECKER : Check packets on queue
237E 6857 :
237E 6858 : Abort all xmit requests on CDB queue
237E 6859 :
57 00C8 C5 D0 237E .1 MOVL UCB$L_XQ_CDB(R5),R7 : Get CDB address
2383 .2 ASSUME CDB_STS_V_INITED EQ 0
DD 0246 C7 E9 2383 .3 BLBC CDB_B_STSTR(R7),10$ : Br if not inited
2388 .4 DSBINT UCB$B_DIPL(R5) : Sync access to CDB
57 00DC C7 9E 238F .5 MOVAB CDB_Q_QUEUES(R7),R7 : Get start of queues
58 01 9A 2394 .6 MOVZBL #CDB_C_ABORTS,R8 : Get number of queues we can abort on
56 57 D0 2397 6867 30$: 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

```

:RNG0001
:RNG0001
-1

:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
:RNG0001
-7

```

53 08 12 23B1 6881 BNEQ 20$ ; Br if no match
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 05 23C0 6887 30$: RSB ; Return to caller
23C1 6888 ;
23C1 6889 ; Subroutine to check for specific cancel
23C1 6890 ;
23C1 6891 CHECKPKT:
OC A3 D5 23C1 6892 TSTL IRPSL_PID(R3) ; Is this an Internal IRP?
11 19 23C4 6893 BLSS 30$ ; 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
OC A3 60 A4 D1 23CE 6900 10$: CMPL PCB$SL_PID(R4),IRPSL_PID(R3) ; PID match?
OE 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$: CMPL PCB$SL_PID(R4),UCB$SL_XQ_PID(R5) ; IS this the starter's PID?
00B8 C5 23DA
28 A3 52 B1 23DD 6905 BNEQ 50$ ; Br if no
05 05 23DF 6906 40$: CMPW R2,IRPSW_CHAN(R3) ; Channel match?
23E3 6907 50$: 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 BSBW CXB_CHECKPKT ; Cancel if appropriate match
2F 12 23EE 6914 BNEQ 20$ ; Br if no match
53 63 0F 23FO 6915 REMQUE (R3),R3 ; Remove from CXB list
23F3 6916 ASSUME CXB$SL_T_IRP EQ CXB$SL_T_UCB
16 24 A3 E8 23F3 6917 BLBS CXB$SL_T_IRP(R3),16$ ; Br if not IRP address -> FAST interface
53 24 A3 D0 23F7 6918 MOVL CXB$SL_T_IRP(R3),R3 ; Else, get IRP address
50 0094 C3 D0 23FB 6919 MOVL IRPSL_XQ_SETUP(R3),R0 ; Get address of SETUP mode buffer
00000000 GF 16 2400 6920 BEQL 13$ ; Br if none
F72E 30 2402 6921 JSB G^COM$DRVDEALMEM ; Else, deallocate the buffer
D7 11 2408 6922 13$: BSBW ABORT_PKT ; Complete the I/O request
54 0192 C5 DD 240B 6923 BRB CXB_CHECKER ; Look for more
50 2C 3C 240D 6924 16$: PUSHL R4 ; Save R4
14 B4 16 240F 6925 MOVL UCB$SL_XQ_FFI(R5),R4 ; Get FFI block address
54 8ED0 11 2414 6926 MOVZWL #SS$ ABORT,R0 ; Set status return
C5 11 2417 6927 JSB @FFISL_XMIT_DONE(R4) ; Complete the XMIT CXB
53 63 D0 241A 6928 POPL R4 ; Restore R4
C3 11 241D 6929 BRB CXB_CHECKER ; Look for more
05 05 241F 6930 20$: MOVL (R3),R3 ; Travel link
2422 6931 BRB 10$ ; Look for more
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$T_IRP(R3),R0      ; Get (presumed) IRP address
      2429 6938      ASSUME  CXB$T_IRP EQ CXB$T_UCB
23 50 E8 2429 6939      BLBS   R0,80$              ; Br if not an IRP address
OC A0 D5 242C 6940      TSTL   IRP$PID(R0)         ; Is this an Internal IRP?
      11 19 242F 6941      BLSS   30$                 ; 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
OC A0 60 A4 D1 2439 6948 10$:  CMPL   PCB$PID(R4),IRP$PID(R0) ; PID match?
      OE 12 243E 6949      BNEQ   50$                 ; Br if no
      08 11 2440 6950      BRB    40$                 ; Try CHAN match
      2442 6951
      60 A4 D1 2442 6952 30$:  CMPL   PCB$PID(R4),UCB$XQ_PID(R5) ; IS this the starter's PID?
00B8 C5 2445
      04 12 2448 6953      BNEQ   50$                 ; Br if no
28 A0 52 B1 244A 6954 40$:  CMPW   R2,IRP$W_CHAN(R0)      ; Channel match?
      05 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 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:
51 00C4 C5 DO 2457 6982 MOVL UCBSL_XQ_DEFUSR(R5),R1 : Try to find shared user
: 04 13 245C 6983 BEQL 10$ : Get address of default user
: 1C 10 245E 6984 BSBB 90$ : Br if no default user
: 19 13 2460 6985 BEQL 40$ : Check for match
50 0098 C5 9E 2462 6986 10$: MOVAB UCBSQ_XQ_SHARE(R5),R0 : Save address of listhead
51 50 DO 2467 6987 MOVL R0,R1 : Copy listhead address
: 246A 6988 ASSUME SHR_L_QFL EQ 0
: 51 61 DO 246A 6989 20$: 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 DO 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
: OC A1 D5 2480 7014 TSTL SHR_L_PID(R1) : Zero PID?
: OD 12 2483 7015 BNEQ 140$ : Br if not
: 07 11 2485 7016 BRB 110$ : Try for CHAN
OC A1 60 A4 D1 2487 7017 100$: CMPL PCB$SL_PID(R4),SHR_L_PID(R1) : PIDs match?
: 04 12 248C 7018 BNEQ 140$ : Br if no - try for next
: 10 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 :--
2493 .2
```

:RNG0001

```

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

```

:RNG0001
-2

```

2493 7140 ADD_MULTI: ; Add up all the multicast addresses
2493 7141 PUSHF #M<R3,R4,R6,R7> ; Save registers
2497 .1 MOVL UCBSL_XQ(CDB(R5)),R4 ; Get CDB address
249C 7144 CLRB CDB_B_MLTTBL(R4) ; Reset number of entries
24A0 7145 PUSHQ R4 ; Save CDB and UCB addresses
24A3 7146 MOVCS #0,CDB_G_MLTTBL(R4),#0,- ; Zero the structure
24A8
24A9 7147 #6*MAX_C_MLT,CDB_G_MLTTBL(R4) ;
24AF 7148 POPQ R4 ; Restore CDB and UCB addresses
24B2 7149 MOVZBL #MAX_C_MLT+1,R3 ; Error if 1 more multicast address
24B5 7150 ; than we can handle
24B5 7151 MNEGL #1,R0 ; Assume success
24B8 7152 MOVAB CDB_G_MLTTBL(R4),R6 ; Get address of Multicast table
24BD 7153 MOVL UCBSL_DDB(R5),R7 ; Get DDB address
24C1 7154 MOVL DDBSL_UCB(R7),R7 ; Get 1st UCB address
24C5 7155 10$: MOVL UCBSL_LINK(R7),R7 ; Get next UCB in list
24C9 7156 BEQL 50$ ; Br if no more UCB's
24CB 7157
24CB 7158 ASSUME UCBSV_XQ_INITED EQ 0
24CB 7159 BLBC UCBSW_DEVSTS(R5),10$ ; Br if not inited
24CF 7160 MOVAB UCBSG_XQ_MULTI(R7),R2 ; Get address of Multicast list
24D4 7161 MOVZBL UCBSB_XQ_MULTI(R7),R1 ; Set number addresses for UCB
24D9 7162 20$: TSTL (R2) ; Is this field unused?
24DB 7163 BNEQ 25$ ; Br if no
24DD 7164 TSTW 4(R2) ; Really?
24E0 7165 BEQL 30$ ; Yes - skip it
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
	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	ADDL	#6, R2	30\$:	: 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	CLRL	R0	40\$:	: Return failure
	00DB	8F	BA	2502	7179	POPR	#*M<R3, R4, R6, R7>	50\$:	: 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:
0048 3F  BB 2507 7206      PUSHR    #^M<R0,R1,R2,R3,R4,R5> ; Move the multicast address list
02A6 8F  28 2509 7207      MOVCS    #MAX_C_MLT*6,CDB_G_MLTTBL(R4),CDB_G_MULTI(R4) ; Copy list
025E C4      250D
025D 3F  BA 2513 7208      POPR    #^M<R0,R1,R2,R3,R4,R5> ; Restore registers
025C C4  90 2515 7209      MOVBS    CDB_B_MLTTBL(R4),CDB_B_MULTI(R4) ; Set number of entries
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:
251D 7231     PUSH  #^M<R0,R1,R2,R3,R4,R5> ; Save multicast address list
0048 3F BB 251D 7231     PUSH  #^M<R0,R1,R2,R3,R4,R5> ; Save registers
00EC 8F 28 251D 7232     MOV  #6*MAX_C_MLT,UCB$G_XQ_MULTI(R5),- ; Save multicast addresses
0134 C5 2523 7233
00E9 3F BA 2529 7234     POP  UCB$G_XQ_MLTTBL(R5) ;
00EA C5 90 252B 7235     MOV  UCB$B_XQ_MULTI(R5),- ; Restore registers
00E9 C5 05 252F 7236     MOV  UCB$B_XQ_MLTTBL(R5) ; Save count of multicast addresses
00EA C5 05 2532 7237     RSB ; Return to caller
2533 7238
2533 7239 RES_MULTI:
2533 7240     PUSH  #^M<R0,R1,R2,R3,R4,R5> ; Restore multicast address list
0048 3F BB 2533 7240     PUSH  #^M<R0,R1,R2,R3,R4,R5> ; Save registers
0134 8F 28 2535 7241     MOV  #6*MAX_C_MLT,UCB$G_XQ_MLTTBL(R5),- ; Restore multicast addresses
00EC C5 2539 7242
00EA 3F BA 253C 7243     POP  UCB$G_XQ_MULTI(R5) ;
00EA C5 90 253F 7244     MOV  UCB$B_XQ_MLTTBL(R5),- ; Restore registers
00E9 C5 05 2541 7245     MOV  UCB$B_XQ_MLTTBL(R5),- ; Restore count of multicast addresses
00E9 C5 05 2545 7246     RSB ; Return to caller
2548 7246
  
```

```

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 : Outputs:
2549 7266 :
2549 7267 :     R0 = Status return for request
2549 7268 :
2549 7269 : If no error:
2549 7270 :     R1 = Address of parameter verification table
2549 7271 : If error:
2549 7272 :     R1 = Bad parameter value
2549 7273 :
2549 7274 : All other registers are preserved.
2549 7275 :
2549 7276 :
2549 7277 :--
2549 7278
2549 7279 VALIDATE_P2::
2549 7280     PUSH  #M<R2,R3,R4,R6,R7,R8,R9> : Validate P2 buffer parameters
2549 7281     MOVL  IRP$L_SVAPTE(R3),R6       : Save registers
2549 7282     BNEQ  20$                       : Get system P2 buffer address
2549 7283 10$: BRW  150$                   : Br if a system buffer
2549 7284                                     : Else, leave
2549 7285
2549 7286 20$: MOVL  UCBS$L_XQ_CDB(R5),R4    : Get CDB address
2549 7287     MOVL  P2B_L_POINTER(R6),R6     : Point to start of P2 data
2549 7288     MOVZWL IRP$W_BCNT(R3),R8       : Get size of P2 buffer
2549 7289 :
2549 7290 : Loop to check next parameter in P2 buffer
2549 7291 :
2549 7292 30$:  SUBL  #2,R8                 : Can we get another parameter?
2549 7293     BLSS  10$                       : Br if no - all done
2549 7294     MOVZWL (R6)+,R1                : Get parameter type from P2
2549 7295 :
2549 7296 : *** NOTE - R2 MUST be the very first item on the top of the stack
2549 7297 :
2549 7298     MOVL  (SP),R7                   : Get verification table address
2549 7299 :
2549 7300 : Loop to check P2 buffer parameter to Line parameter table
2549 7301 :
2549 7302     ASSUME PRM W TYPE EQ 0
2549 7303 40$: MOVL  (R7)+,R0               : Get parameter type code
2549 7304     BNEQ  45$                       : Br if NOT end of verify table
2549 7305 43$: BRW  170$                   : Else exit in error

```

:RNG0001
-2

```

56 03DC 8F BB 2549 7280
   2C A3 D0 254D 7281
     03 12 2551 7282
   0121 31 2553 7283
54 00C8 C5 D0 2556 7284
   56 66 D0 255B 7287
58 32 A3 3C 255E 7288
   2562 7289
   2562 7290
58 02 C2 2562 7292
   EC 19 2565 7293
51 86 3C 2567 7294
   256A 7295
   256A 7296
57 6E D0 256A 7297
   256A 7298
   256D 7299
   256D 7300
   256D 7301
50 87 B0 256D 7302
   03 12 2570 7303
   0107 31 2572 7305

```

```

2575 7306
2575 7307 45$: ASSUME PRM_B_FLAG EQ PRM_W_TYPE+2
50 59 87 9A 2575 7308 MOVZBL (R7)+,R9 ; Get flags byte
50 F000 8F AA 2578 7309 BICW #^C<PRM_TYP_M_CODE>,R0 ; Clear all but type code
50 51 B1 257D 7310 CMPW R1,R0 ; Parameters match?
57 17 13 2580 7311 BEQL 50$ ; 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
258B 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$: MOVW (R7)+,R0 ; Get offset + width
52 50 0A EF 259C 7321 EXTZV #PRM_OFF_V_WIDTH,- ; Get width only
50 06 00 EF 259E 7322 EXTZV #PRM_OFF_S_WIDTH,R0,R2 ;
50 50 0A EF 25A1 7323 EXTZV #PRM_OFF_V_VALUE,- ; Get offset only
05 59 03 E0 25A3 7324 #PRM_OFF_S_VALUE,R0,R0 ;
50 55 C0 25A6 7325 BBS #PRM_FLG_V_CDB,R9,55$ ; Br if CDB datum
50 03 11 25AA 7326 ADDL R5,R0 ; Compute offset in UCB
50 54 C0 25AD 7327 BRB 57$ ; Continue
50 54 C0 25AF 7328 55$: ADDL R4,R0 ; Compute offset in CDB
31 FB A7 0C E0 25B2 7329 57$: ASSUME PRM_B_FLAG EQ PRM_W_TYPE+2
58 04 C2 25B2 7330 BBS #PRM_TYP_V_STRING,-5(R7),95$ ; Br if string parameter
53 86 D0 25B7 7331 SUBL #4,R8 ; Must be longword value
25BA 7332 BLSS 43$ ; Br if error
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
25D2 7348 ;
25D3 7349 ; Longword value
25D3 7350 ;
60 53 D1 25D3 7351 80$: CMPL R3,(R0) ; Is this the same?
OBOE 8F 6E 12 25D6 7352 90$: BNEQ 100$ ; Br if no - continue checks
51 B1 25D8 7353 CMPW R1,#NMASC_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$: SUBL #2,R8 ; Can we fetch string length?
F8 19 25EB 7362 BLSS 95$ ; 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?
FO 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 OB21 8F B1 25FD 7369 CMPW #NMASC_PCLI_DES,R1 ; Is this the destination address?
07 13 2602 7370 BEQL 96$ ; Br if yes
51 OB04 8F B1 2604 7371 CMPW #NMASC_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 OB0F 8F B1 2613 7376 97$: CMPW #NMASC_PCLI_MCA,R1 ; Is this the multicast address list?
3E 12 2618 7377 BNEQ 130$ ; 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 UCBSB FIPL(R5) ; Sync access to UCB
FEF3 30 2627 7381 BSBW SAV_MULTI ; Save the multicast addresses
0240 8F BB 262A 7382 PUSHR #MZR6,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 POPR #M<R6,R9> ; R0 = return status
0240 8F BA 2637 7387 BSBW RES_MULTI ; Restore registers
FEF5 30 263B 7388 ENBINT ; Restore the multicast list
263E 7389 BLBC R0,170$ ; Restore IPL
38 50 E9 2641 7390 BRB 130$ ; 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
58 A5 52 B3 2663 7402 BITW R2,UCBSW_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

```

:RNG0001
-2

```

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:
2684 7442      PUSH  R1,R2,R3,R4,R6,R7,R8,R9,R10 ; Change the UCB/CDB parameters
2684 7443      MOV   R2,R10 ; Save registers
2684 7444      MOV   IRP$L_SVAPTE(R3),R6 ; Save table address
2684 7445      BNEQ  5$ ; Get system P2 buffer address
2684 7446      BRW  120$ ; Br if system buffer
2684 7447      ; Else, return
2684 7448
2684 7449  5$:
2684 7450      MOV   UCBS$L_XQ_CDB(R5),R4 ; Get CDB address
2684 7451      MOV   P2B$L_POINTER(R6),R6 ; Point to start of data
2684 7452      MOVZWL IRP$W_BCNT(R3),R8 ; Get size of P2 buffer
2684 7453      :
2684 7454      : Loop to get next parameter from P2 buffer
2684 7455  10$:
2684 7456      SUBL  #2,R8 ; Try to get next parameter
2684 7457      BLSS  3$ ; Br if not there
2684 7458      MOVZWL (R6)+,R0 ; Get parameter type from P2
2684 7459      BEQL  90$ ; Br if null value parameter
2684 7460      MOV   R10,R7 ; Get verification table address
2684 7461      CMPW  R0,#NMASC_PCLI_PTY ; Is this the protocol type?
2684 7462      BNEQ  20$ ; Br if not
2684 7463      BISW  #UCBSM_XQ_PROTYP,- ; Indicate that protocol type specified
2684 7464      UCBSW_DEVSTS(R5)
2684 7465      :
2684 7466      : Loop to store buffer parameter in UCB/CDB
2684 7467      :
2684 7468  20$:
2684 7469      ASSUME PRM_W_TYPE EQ 0
2684 7470      MOVZWL (R7)+,R1 ; Get parameter type code
2684 7471      BEQL  90$ ; Br if end of verify table
2684 7472      BICW  #^C<PRM_TYP_M_CODE>,R1 ; Clear all but type code
2684 7473      ASSUME PRM_B_FLAG EQ PRM_W_TYPE+2
2684 7474      MOVZBL (R7)+,R9 ; Get flags byte
2684 7475      CMPW  R0,R1 ; Parameters match?
2684 7476      BEQL  30$ ; Br if yes

```

```

07DE 8F BB
5A 52 DO
56 2C A3 DO
03 12
00D8 31
54 00C8 C5 DO
56 66 DO
58 32 A3 3C
26A0
26A0
58 02 C2
EC 19
50 86 3C
79 13
57 5A DO
OBOE 8F 50 B1
04 12
04 A8
68 A5
51 87 3C
66 13
51 F000 8F AA
26C2
59 87 9A
51 50 B1
17 13

```

```

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$: MOVW (R7)+,R1 ; Get offset + width
      52 51 0A EF 26E4 7484 EXTZV #PRM_OFF_V_WIDTH,- ; Get width only
      51 51 00 EF 26E6 7485 #PRM_OFF_S_WIDTH,R1,R2 ;
      51 51 0A EF 26E9 7486 EXTZV #PRM_OFF_V_VALUE,- ; Get offset only
      05 59 03 E0 26EB 7487 #PRM_OFF_S_VALUE,R1,R1 ;
      51 55 C0 26EE 7488 BBS #PRM_FLG_V_CDB,R9,40$ ; Br if CDB datum
      51 03 11 26F2 7489 ADDL R5,RT ; Compute offset in UCB
      51 54 C0 26F5 7490 BRB 50$ ; Continue
      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 #NMASC_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$: CMPW #NMASC_PCLI_PHA,R0 ; Is this the physical address?
      05 12 274B 7530 BNEQ 103$ ; Br if no
      022D 30 274D 7531 BSBW SET_PHYAD ; Else set new physical address

```

```
50  OBOF 14 11 2750 7532 BRB 110$ ; Continue
      8F B1 2752 7533 103$: CMPW #NMASC_PCLI_MCA,R0 ; Is this the multicast address list?
      05 12 2757 7534 BNEQ 105$ ; 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$: PUSHR #*M<R1,R2,R3,R4,R5> ; Save registers
61  66 59 28 2760 7538 MOV3 R9,(R6),(R1) ; Store string
      3E BA 2764 7539 POPR #*M<R1,R2,R3,R4,R5> ; Restore registers
      56 59 C0 2766 7540 110$: 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::
2771 7571     PUSH  R1,R2,R3,R4,R6,R7,R8,R9 ; Return P2 buffer parameters
2775 7572     CLRL  R0 ; Save registers
2777 7573     MOVL  IRP$L_XQ_P2BUF(R3),R6 ; Assume no P2 buffer given
277B 7574     BNEQ  5$ ; Get user P2 buffer address
277D 7575     BRW   70$ ; Br if given
2780 7576     ; Else, return
2780 7577
2780 7578 5$:
2780 7579     MOVL  UCBS$L_XQ_CDB(R5),R4 ; Get CDB address
2785 7580     MOVZWL IRP$W_XQ_USERSIZ(R3),R8 ; Get size of user buffer
2789 7581     PUSH  R6 ; Save start of data address
278B 7582     PUSH  R3 ; Save IRP address
278D 7583     MOVAB LINE_PARAM,R1 ; Get address of verification table
2792 7584
2792 7585 : Loop to return next parameter
2792 7586 :
2792 7587 10$:
2792 7588     ASSUME PRM_W_TYPE EQ 0
2795 7589     MOVW  (R1)+,R7 ; Get parameter type code
2797 7590     BNEQ  11$ ; Br if end of verify table
279A 7591     BRW   65$ ; ...
279A 7592
279A 7593 11$:
279A 7594     BICW3 #^C<PRM_TYP_M_CODE>,R7,R9 ; Get only the type code
279F 7595
279F 7596     MOVZBL (R1)+,R3 ; Get flags byte
27A0 7597     MOVW  (R1)+,R0 ; Get offset + width
27A3 7598
27A6 7599 : We will only return NMASC_PCLI_DES to the SHARED-LIMITED users.
27A6 7600 :
27A6 7601 :
27A6 7602
27A6 7603
27A6 7604
27A6 7605
27A6 7606
27A6 7607
27A6 7608
27A6 7609
27A6 7610
27A6 7611
27A6 7612
27A6 7613
27A6 7614
27A6 7615
27A6 7616
27A6 7617
27A6 7618
27A6 7619
27A6 7620
27A6 7621
27A6 7622
27A6 7623
27A6 7624
27A6 7625
27A6 7626
27A6 7627
27A6 7628
27A6 7629
27A6 7630
27A6 7631
27A6 7632
27A6 7633
27A6 7634
27A6 7635
27A6 7636
27A6 7637
27A6 7638
27A6 7639
27A6 7640
27A6 7641
27A6 7642
27A6 7643
27A6 7644
27A6 7645
27A6 7646
27A6 7647
27A6 7648
27A6 7649
27A6 7650
27A6 7651
27A6 7652
27A6 7653
27A6 7654
27A6 7655
27A6 7656
27A6 7657
27A6 7658
27A6 7659
27A6 7660
27A6 7661
27A6 7662
27A6 7663
27A6 7664
27A6 7665
27A6 7666
27A6 7667
27A6 7668
27A6 7669
27A6 7670
27A6 7671
27A6 7672
27A6 7673
27A6 7674
27A6 7675
27A6 7676
27A6 7677
27A6 7678
27A6 7679
27A6 7680
27A6 7681
27A6 7682
27A6 7683
27A6 7684
27A6 7685
27A6 7686
27A6 7687
27A6 7688
27A6 7689
27A6 7690
27A6 7691
27A6 7692
27A6 7693
27A6 7694
27A6 7695
27A6 7696
27A6 7697
27A6 7698
27A6 7699
27A6 7700
27A6 7701
27A6 7702
27A6 7703
27A6 7704
27A6 7705
27A6 7706
27A6 7707
27A6 7708
27A6 7709
27A6 7710
27A6 7711
27A6 7712
27A6 7713
27A6 7714
27A6 7715
27A6 7716
27A6 7717
27A6 7718
27A6 7719
27A6 7720
27A6 7721
27A6 7722
27A6 7723
27A6 7724
27A6 7725
27A6 7726
27A6 7727
27A6 7728
27A6 7729
27A6 7730
27A6 7731
27A6 7732
27A6 7733
27A6 7734
27A6 7735
27A6 7736
27A6 7737
27A6 7738
27A6 7739
27A6 7740
27A6 7741
27A6 7742
27A6 7743
27A6 7744
27A6 7745
27A6 7746
27A6 7747
27A6 7748
27A6 7749
27A6 7750
27A6 7751
27A6 7752
27A6 7753
27A6 7754
27A6 7755
27A6 7756
27A6 7757
27A6 7758
27A6 7759
27A6 7760
27A6 7761
27A6 7762
27A6 7763
27A6 7764
27A6 7765
27A6 7766
27A6 7767
27A6 7768
27A6 7769
27A6 7770
27A6 7771
27A6 7772
27A6 7773
27A6 7774
27A6 7775
27A6 7776
27A6 7777
27A6 7778
27A6 7779
27A6 7780
27A6 7781
27A6 7782
27A6 7783
27A6 7784
27A6 7785
27A6 7786
27A6 7787
27A6 7788
27A6 7789
27A6 7790
27A6 7791
27A6 7792
27A6 7793
27A6 7794
27A6 7795
27A6 7796
27A6 7797
27A6 7798
27A6 7799
27A6 7800
27A6 7801
27A6 7802
27A6 7803
27A6 7804
27A6 7805
27A6 7806
27A6 7807
27A6 7808
27A6 7809
27A6 7810
27A6 7811
27A6 7812
27A6 7813
27A6 7814
27A6 7815
27A6 7816
27A6 7817
27A6 7818
27A6 7819
27A6 7820
27A6 7821
27A6 7822
27A6 7823
27A6 7824
27A6 7825
27A6 7826
27A6 7827
27A6 7828
27A6 7829
27A6 7830
27A6 7831
27A6 7832
27A6 7833
27A6 7834
27A6 7835
27A6 7836
27A6 7837
27A6 7838
27A6 7839
27A6 7840
27A6 7841
27A6 7842
27A6 7843
27A6 7844
27A6 7845
27A6 7846
27A6 7847
27A6 7848
27A6 7849
27A6 7850
27A6 7851
27A6 7852
27A6 7853
27A6 7854
27A6 7855
27A6 7856
27A6 7857
27A6 7858
27A6 7859
27A6 7860
27A6 7861
27A6 7862
27A6 7863
27A6 7864
27A6 7865
27A6 7866
27A6 7867
27A6 7868
27A6 7869
27A6 7870
27A6 7871
27A6 7872
27A6 7873
27A6 7874
27A6 7875
27A6 7876
27A6 7877
27A6 7878
27A6 7879
27A6 7880
27A6 7881
27A6 7882
27A6 7883
27A6 7884
27A6 7885
27A6 7886
27A6 7887
27A6 7888
27A6 7889
27A6 7890
27A6 7891
27A6 7892
27A6 7893
27A6 7894
27A6 7895
27A6 7896
27A6 7897
27A6 7898
27A6 7899
27A6 7900
27A6 7901
27A6 7902
27A6 7903
27A6 7904
27A6 7905
27A6 7906
27A6 7907
27A6 7908
27A6 7909
27A6 7910
27A6 7911
27A6 7912
27A6 7913
27A6 7914
27A6 7915
27A6 7916
27A6 7917
27A6 7918
27A6 7919
27A6 7920
27A6 7921
27A6 7922
27A6 7923
27A6 7924
27A6 7925
27A6 7926
27A6 7927
27A6 7928
27A6 7929
27A6 7930
27A6 7931
27A6 7932
27A6 7933
27A6 7934
27A6 7935
27A6 7936
27A6 7937
27A6 7938
27A6 7939
27A6 7940
27A6 7941
27A6 7942
27A6 7943
27A6 7944
27A6 7945
27A6 7946
27A6 7947
27A6 7948
27A6 7949
27A6 7950
27A6 7951
27A6 7952
27A6 7953
27A6 7954
27A6 7955
27A6 7956
27A6 7957
27A6 7958
27A6 7959
27A6 7960
27A6 7961
27A6 7962
27A6 7963
27A6 7964
27A6 7965
27A6 7966
27A6 7967
27A6 7968
27A6 7969
27A6 7970
27A6 7971
27A6 7972
27A6 7973
27A6 7974
27A6 7975
27A6 7976
27A6 7977
27A6 7978
27A6 7979
27A6 7980
27A6 7981
27A6 7982
27A6 7983
27A6 7984
27A6 7985
27A6 7986
27A6 7987
27A6 7988
27A6 7989
27A6 7990
27A6 7991
27A6 7992
27A6 7993
27A6 7994
27A6 7995
27A6 7996
27A6 7997
27A6 7998
27A6 7999
27A6 8000
27A6 8001
27A6 8002
27A6 8003
27A6 8004
27A6 8005
27A6 8006
27A6 8007
27A6 8008
27A6 8009
27A6 8010
27A6 8011
27A6 8012
27A6 8013
27A6 8014
27A6 8015
27A6 8016
27A6 8017
27A6 8018
27A6 8019
27A6 8020
27A6 8021
27A6 8022
27A6 8023
27A6 8024
27A6 8025
27A6 8026
27A6 8027
27A6 8028
27A6 8029
27A6 8030
27A6 8031
27A6 8032
27A6 8033
27A6 8034
27A6 8035
27A6 8036
27A6 8037
27A6 8038
27A6 8039
27A6 8040
27A6 8041
27A6 8042
27A6 8043
27A6 8044
27A6 8045
27A6 8046
27A6 8047
27A6 8048
27A6 8049
27A6 8050
27A6 8051
27A6 8052
27A6 8053
27A6 8054
27A6 8055
27A6 8056
27A6 8057
27A6 8058
27A6 8059
27A6 8060
27A6 8061
27A6 8062
27A6 8063
27A6 8064
27A6 8065
27A6 8066
27A6 8067
27A6 8068
27A6 8069
27A6 8070
27A6 8071
27A6 8072
27A6 8073
27A6 8074
27A6 8075
27A6 8076
27A6 8077
27A6 8078
27A6 8079
27A6 8080
27A6 8081
27A6 8082
27A6 8083
27A6 8084
27A6 8085
27A6 8086
27A6 8087
27A6 8088
27A6 8089
27A6 8090
27A6 8091
27A6 8092
27A6 8093
27A6 8094
27A6 8095
27A6 8096
27A6 8097
27A6 8098
27A6 8099
27A6 8100
27A6 8101
27A6 8102
27A6 8103
27A6 8104
27A6 8105
27A6 8106
27A6 8107
27A6 8108
27A6 8109
27A6 8110
27A6 8111
27A6 8112
27A6 8113
27A6 8114
27A6 8115
27A6 8116
27A6 8117
27A6 8118
27A6 8119
27A6 8120
27A6 8121
27A6 8122
27A6 8123
27A6 8124
27A6 8125
27A6 8126
27A6 8127
27A6 8128
27A6 8129
27A6 8130
27A6 8131
27A6 8132
27A6 8133
27A6 8134
27A6 8135
27A6 8136
27A6 8137
27A6 8138
27A6 8139
27A6 8140
27A6 8141
27A6 8142
27A6 8143
27A6 8144
27A6 8145
27A6 8146
27A6 8147
27A6 8148
27A6 8149
27A6 8150
27A6 8151
27A6 8152
27A6 8153
27A6 8154
27A6 8155
27A6 8156
27A6 8157
27A6 8158
27A6 8159
27A6 8160
27A6 8161
27A6 8162
27A6 8163
27A6 8164
27A6 8165
27A6 8166
27A6 8167
27A6 8168
27A6 8169
27A6 8170
27A6 8171
27A6 8172
27A6 8173
27A6 8174
27A6 8175
27A6 8176
27A6 8177
27A6 8178
27A6 8179
27A6 8180
27A6 8181
27A6 8182
27A6 8183
27A6 8184
27A6 8185
27A6 8186
27A6 8187
27A6 8188
27A6 8189
27A6 8190
27A6 8191
27A6 8192
27A6 8193
27A6 8194
27A6 8195
27A6 8196
27A6 8197
27A6 8198
27A6 8199
27A6 8200
27A6 8201
27A6 8202
27A6 8203
27A6 8204
27A6 8205
27A6 8206
27A6 8207
27A6 8208
27A6 8209
27A6 8210
27A6 8211
27A6 8212
27A6 8213
27A6 8214
27A6 8215
27A6 8216
27A6 8217
27A6 8218
27A6 8219
27A6 8220
27A6 8221
27A6 8222
27A6 8223
27A6 8224
27A6 8225
27A6 8226
27A6 8227
27A6 8228
27A6 8229
27A6 8230
27A6 8231
27A6 8232
27A6 8233
27A6 8234
27A6 8235
27A6 8236
27A6 8237
27A6 8238
27A6 8239
27A6 8240
27A6 8241
27A6 8242
27A6 8243
27A6 8244
27A6 8245
27A6 8246
27A6 8247
27A6 8248
27A6 8249
27A6 8250
27A6 8251
27A6 8252
27A6 8253
27A6 8254
27A6 8255
27A6 8256
27A6 8257
27A6 8258
27A6 8259
27A6 8260
27A6 8261
27A6 8262
27A6 8263
27A6 8264
27A6 8265
27A6 8266
27A6 8267
27A6 8268
27A6 8269
27A6 8270
27A6 8271
27A6 8272
27A6 8273
27A6 8274
27A6 8275
27A6 8276
27A6 8277
27A6 8278
27A6 8279
27A6 8280
27A6 8281
27A6 8282
27A6 8283
27A6 8284
27A6 8285
27A6 8286
27A6 8287
27A6 8288
27A6 8289
27A6 8290
27A6 8291
27A6 8292
27A6 8293
27A6 8294
27A6 8295
27A6 8296
27A6 8297
27A6 8298
27A6 8299
27A6 8300
27A6 8301
27A6 8302
27A6 8303
27A6 8304
27A6 8305
27A6 8306
27A6 8307
27A6 8308
27A6 8309
27A6 8310
27A6 8311
27A6 8312
27A6 8313
27A6 8314
27A6 8315
27A6 8316
27A6 8317
27A6 8318
27A6 8319
27A6 8320
27A6 8321
27A6 8322
27A6 8323
27A6 8324
27A6 8325
27A6 8326
27A6 8327
27A6 8328
27A6 8329
27A6 8330
27A6 8331
27A6 8332
27A6 8333
27A6 8334
27A6 8335
27A6 8336
27A6 8337
27A6 8338
27A6 8339
27A6 8340
27A6 8341
27A6 8342
27A6 8343
27A6 8344
27A6 8345
27A6 8346
27A6 8347
27A6 8348
27A6 8349
27A6 8350
27A6 8351
27A6 8352
27A6 8353
27A6 8354
27A6 8355
27A6 8356
27A6 8357
27A6 8358
27A6 8359
27A6 8360
27A6 8361
27A6 8362
27A6 8363
27A6 8364
27A6 8365
27A6 8366
27A6 8367
27A6 8368
27A6 8369
27A6 8370
27A6 8371
27A6 8372
27A6 8373
27A6 8374
27A6 8375
27A6 8376
27A6 8377
27A6 8378
27A6 8379
27A6 8380
27A6 8381
27A6 8382
27A6 8383
27A6 8384
27A6 8385
27A6 8386
27A6 8387
27A6 8388
27A6 8389
27A6 8390
27A6 8391
27A6 8392
27A6 8393
27A6 8394
27A6 8395
27A6 8396
27A6 8397
27A6 8398
27A6 8399
27A6 8400
27A6 8401
27A6 8402
27A6 8403
27A6 8404
27A6 8405
27A6 8406
27A6 8407
27A6 8408
27A6 8409
27A6 8410
27A6 8411
27A6 8412
27A6 8413
27A6 8414
27A6 8415
27A6 8416
27A6 8417
27A6 8418
27A6 8419
27A6 8420
27A6 8421
27A6 8422
27A6 8423
27A6 8424
27A6 8425
27A6 8426
27A6 8427
27A6 8428
27A6 8429
27A6 8430
27A6 8431
27A6 8432
27A6 8433
27A6 8434
27A6 8435
27A6 8436
27A6 8437
27A6 8438
27A6 8439
27A6 8440
27A6 8441
27A6 8442
27A6 8443
27A6 8444
27A6 8445
27A6 8446
27A6 8447
27A6 8448
27A6 8449
27A6 8450
27A6 8451
27A6 8452
27A6 8453
27A6 8454
27A6 8455
27A6 8456
27A6 8457
27A6 8458
27A6 8459
27A6 8460
27A6 8461
27A6 8462
27A6 8463
27A6 8464
27A6 8465
27A6 8466
27A6 8467
27A6 8468
27A6 8469
27A6 8470
27A6 8471
27A6 8472
27A6 8473
27A6 8474
27A6 8475
27A6 8476
27A6 8477
27A6 8478
27A6 8479
27A6 8480
27A6 8481
27A6 8482
27A6 8483
27A6 8484
27A6 8485
27A6 8486
27A6 8487
27A6 8488
27A6 8489
27A6 8490
27A6 8491
27A6 8492
27A6 8493
27A6 8494
27A6 8495
27A6 8496
27A6 8497
27A6 8498
27A6 8499
27A6 8500
27A6 8501
27A6 8502
27A6 8503
27A6 8504
27A6 8505
27A6 8506
27A6 8507
27A6 8508
27A6 8509
27A6 8510
27A6 8511
27A6 8512
27A6 8513
27A6 8514
27A6 8515
27A6 8516
27A6 8517
27A6 8518
27A6 8519
27A6 8520
27A6 8521
27A6 8522
27A6 8523
27A6 8524
27A6 8525
27A6 8526
27A6 8527
27A6 8528
27A6 8529
27A6 8530
27A6 8531
27A6 8532
27A6 8533
27A6 8534
27A6 8535
27A6 8536
27A6 8537
27A6 8538
27A6 8539
27A6 8540
27A6 8541
27A6 8542
27A6 8543
27A6 8544
27A6 8545
27A6 8546
27A6 8547
27A6 8548
27A6 8549
27A6 8550
27A6 8551
27A6 8552
27A6 8553
27A6 8554
27A6 8555
27A6 8556
27A6 8557
27A6 8558
27A6 8559
27A6 8560
27A6 8561
27A6 8562
27A6 8563
27A6 8564
27A6 8565
27A6 8566
27A6 8567
27A6 8568
27A6 8569
27A6 8570
27A6 8571
27A6 8572
27A6 8573
27A6 8574
27A6 8575
27A6 8576
27A6 8577
27A6 8578
27A6 8579
27A6 8580
27A6 8581
27A6 8582
27A6 8583
27A6 8584
27A6 8585
27A6 8586
27A6 8587
27A6 8588
27A6 8589
27A6 8590
27A6 8591
27A6 8592
27A6 8593
27A6 8594
27A6 8595
27A6 8596
27A6 8597
27A6 8598
27A6 8599
27A6 8600
27A6 8601
27A6 8602
27A6 8603
27A6 8604
27A6 8605
27A6 8606
27A6 8607
27A6 8608
27A6 8609
27A6 8610
27A6 8611
27A6 8612
27A6 8613
27A6 8614
27A6 8615
27A6 8616
27A6 8617
27A6 8618
27A6 8619
27A6 8620
27A6 8621
27A6 8622
27A6 8623
27A6 8624
27A6 8625
27A6 8626
27A6 8627
27A6 8628
27A6 8629
27A6 8630
27A6 8631
27A6 8632
27A6 8633
27A6 8634
27A6 8635
27A6 8636
27A6 8637
27A6 8638
27A6 8639
27A6 8640
27A6 8641
27A6 8642
27A6 8643
27A6 8644
27A6 8645
27A6 8646
27A6 8647
27A6 8648
27A6 8649
27A6 8650
27A6 8651
27A6 8652
27A6 8653
27A6 8654
27A6 8655
27A6 8656
27A6 8657
27A6 8658
27A6 8659
27A6 8660
27A6 8661
27A6 8662
27A6 8663
27A6 8664
27A6 8665
27A6 8666
27A6 8667
27A6 8668
27A6 8669
27A6 8670
27A6 8671
27A6 8672
27A6 8673
27A6 8674
27A6 8675
27A6 8676
27A6 8677
27A6 8678
27A6 8679
27A6 8680
27A6 8681
27A6 8682
27A6 8683
27A6 8684
27A6 8685
27A6 8686
27A6 8687
27A6 8688
27A6 8689
27A6 8690
27A6 8691
27A6 8692
27A6 8693
27A6 8694
27A6 8695
27A6 8696
27A6 8697
27A6 8698
27A6 8699
27A6 8700
27A6 8701
27A6 8702
27A6 8703
27A6 870
```



```

50 50 00 EF 27B9 7603 EXTZV #PRM_OFF_V_VALUE,- : Get offset only
05 50 0A 27BB 7604 #PRM_OFF_S_VALUE,R0,R0 :
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$: CMPW #NMASC_PCLI_MCA,R9 : Is this the multicast address list?
05 12 2810 7637 BNEQ 57$ : 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$: SUBL #8,R8 : Any room left?
08 19 281A 7641 BLSS 60$ : 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

```



```

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:
004C 8F BB 2878 7737 PUSH  #M<R2,R3,R6> ; Validate the physical address
50 01 CE 287C 7738 MNEGL #1,R0 ; Save some registers
56 53 C2 287F 7739 SUBL R3,R6 ; Assume success
53 02 C2 2882 7740 SUBL #2,R3 ; Point back at start of list
22 19 2885 7741 BLSS 30$ ; Can we read modifier word?
2887 7742 ; Br if no - error
2887 7743 :
2887 7744 : Make sure modifier word is valid.
2887 7745 :
2887 7746 :
2887 7747 ASSUME NMASC_LINMC_SET EQ 1
2887 7748 ASSUME NMASC_LINMC_CLR EQ 2
2887 7749 ASSUME NMASC_LINMC_CAL EQ 3
52 86 3C 2887 7750 ASSUME NMASC_LINMC_SDF EQ 4
2887 7751 MOVZWL (R6)+,R2 ; Get modifier value
288A 7752 $DISPATCH R2,TYPE=B,- ; Dispatch on modifier value
288A 7753 <-
288A 7754 <NMASC_LINMC_SET 20$>,- ; Set the address
288A 7755 <NMASC_LINMC_CLR 40$>,- ; Clear the address
288A 7756 <NMASC_LINMC_CAL 30$>,- ; 3 - invalid value
288A 7757 <NMASC_LINMC_SDF 10$>,- ; 4 - check it out more
288A 7758 >
2896 7759 BRB 30$ ; Any other values are invalid
51 0B04 8F B1 2898 7761 10$: CMPW #NMASC_PCLI_PHA,R1 ; Set to def physical addr requested?
0A 12 289D 7762 BNEQ 30$ ; Return failure if not
0A 11 289F 7763 BRB 40$ ; Else, success
53 06 D1 28A1 7764 ;
03 12 28A4 7765 20$: CMPL #6,R3 ; Is string size okay?
02 66 E9 28A6 7766 BNEQ 30$ ; Br if not
BLBC (R6),40$ ; Br if a physical address
  
```



```

      .SBTIL MATCH_MULTI - CHECK MULTICAST ADDRESS
2A0F 8052
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 promiscuous 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:
2A0F 8078     PUSHQ   R2           ; Find multicast address in UCB
      50     D4     2A12 8079     CLRL    R0           ; Save R2, R3
2A14 8080           ; Assume failure
      22     68     A5     E9     2A14 8081     ASSUME  UCBSV_XQ INITED EQ 0
2A18 8082           ; Br if PROTOCOL TYPE is not valid
      50     D6     2A18 8083     INCL    R0           ; Assume success
2A1A 8084           ;
2A1A 8085           ;
      1B     00DE  C5     E9     2A1A 8086     BLBC   UCBSB_XQ_PRM(R5),10$ ; Br if promiscuous mode - OKAY
      16     00DF  C5     E9     2A1F 8087     BLBC   UCBSB_XQ_MLT(R5),10$ ; Br if all multicasts are enabled
      51     38     A2     7D     2A24 8088     MOVQ   CXBSG_R_DEST(R2),R1 ; Get multicast address
      FFFFFFFF 8F     D1     2A28 8089     CMPL   #BRDCSTT,R1       ; Is this the broadcast address?
      51
      07     12     2A2E 8090           ;
      52     FFFF  8F     B1     2A31 8091     BNEQ   5$           ; Br if no
      02     13     2A36 8092     CMPW   #BRDCST2,R2    ; Really?
      04     10     2A38 8093     BEQL   10$          ; Br if yes - everybody get this one
      5$:    10$:  2A3A 8094     BSBB   MATCH_ADDRESS ; Else, find exact match
      10$:   2A3D 8095     POPQ   R2           ; Restore R2, R3
      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:
2A3E 8123      PUSHL   R4                ; Find multicast address in UCB
2A3E 8124      MOVZBL  S^#SS$ NORMAL,R0    ; Save R4
2A3E 8125      MOVZBL  UCBSB_XQ_MULTI(R5),R4 ; Assume success
2A3E 8126      MOVAB   UCBSG_XQ_MULTI(R5),R3 ; Set number of multicast addresses
2A3E 8127      10$:    CMPL   (R3)+,R1      ; Point to start of multicast lists
2A3E 8128      BNEQ   20$                ; Is this a match?
2A3E 8129      CMPW   (R3),R2            ; Br if no - skip to next
2A3E 8130      BEQL   30$                ; Is it really?
2A3E 8131      20$:    ADDL   #2,R3        ; Br if yes - all done
2A3E 8132      SOBGTR R4,10$            ; Skip to next entry
2A3E 8133
2A3E 8134      30$:    CLRL   R0                ; Br if more in list
2A3E 8135      SUBL   #4,R3            ; Return failure
2A3E 8136      POPL   R4                ; Backup pointer
2A3E 8137      RSB                    ; Restore R4
                                ; Return to caller

```

```

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:
2A66 8161      DSBINT  UCB$B_FIPL(R5)      : Poke user process
2A66 8162      CLRL   -(SP)              : Sync access to UCB
2A66 8163      MOVAB  UCB$L_XQ_AST(R5),R1 : Assume failure
2A66 8164      TSTL   (R1)              : Get AST listhead
2A66 8165      BEQL   30$              : Empty?
2A66 8166      INCL   (SP)              : Branch if yes
2A66 8167      PUSHL  R4                : Indicate success
2A66 8168      MOVL   R1,R4            : Save R4
2A66 8169      MOVL   (R1),R1         : Copy listhead address
2A66 8170      BEQL   20$              : Address a block
2A66 8171      MOVL   UCB$L_DEVDEPEND(R5),- : Branch if done
2A66 8172      ACB$L_KAST+4(R1)        : Change parameter
2A66 8173      BRB    10$              : return status
2A66 8174      JSB    G^COM$DELATTNAST : Continue thru AST blocks
2A66 8175      POPL   R4                : Deliver the AST's
2A66 8176      BRB    10$              : Restore R4
2A66 8177      POPL   R0                : Return success indicator
2A66 8178      ENBINT
2A66 8179      RSB

```

```

51 00C0 7E D4 2A6D 8162
    61 D5 2A6F 8163
    1C 13 2A74 8164
    6E D6 2A76 8165
    54 DD 2A78 8166
54 51 D0 2A7A 8167
51 61 D0 2A7C 8168
    07 13 2A7E 8169
    44 A5 D0 2A82 8170
    1C A1 2A84 8171
    F4 11 2A87 8172
00000000 GF 16 2A89 8173
    54 8ED0 2A8B 8174
    50 8ED0 2A91 8175
    2A94 8176
    2A94 8177
    2A97 8178
    05 2A9A 8179

```


XQDRIVER
Symbol table

- VAX/VMS QNA driver

G 5

8-JAN-1985 17:49:06 VAX/VMS Macro V04-00 Page 179
5-SEP-1984 00:20:54 [DRIVER.BUGSRC]XQDRIVER.MAR;1 (81)

\$\$\$	= 00000020	R	02	CDB_B_NEXTXMT	00000018	G
\$\$\$FLG	= 00000013			CDB_B_PRM	00000247	G
\$\$\$MAP	= 00000000			CDB_B_RVCNT	00000106	G
\$\$\$OFF	= 000000E8			CDB_B_RCVMAP	0000001A	G
\$\$\$SIZ	= 00000008			CDB_B_SETPRM	00000249	G
\$\$\$TYP	= 00000429			CDB_B_SPARE	00000118	G
\$\$\$WID	= 00000002			CDB_B_STS	00000246	G
\$\$BASE	= 00000001			CDB_B_TIM_XMT	0000020A	G
\$\$DISPL	= 00000009			CDB_B_TYPE	0000000A	G
\$\$GENSW	= 00000001			CDB_B_UNTCNT	0000020B	G
\$\$HIGH	= 00000008			CDB_B_XMTCNT	00000107	G
\$\$LIMIT	= 00000007			CDB_B_XMTMAP	0000001B	G
\$\$LOW	= 00000001			CDB_C_ABORTS	= 00000001	
\$\$MNSW	= 00000001			CDB_C_LENGTH	000002F0	G
\$\$MXSW	= 00000001			CDB_C_MAPPED	= 000000A8	
\$\$OP	= 00000002			CDB_C_QUEUES	= 00000005	
ABORTIO	00000340	R	03	CDB_C_SETPRM	= 00000001	
ABORTIO_BR	00000288	R	03	CDB_C_ZERO	0000020A	G
ABORT_IRP	00000CC3	R	03	CDB_G_COUNTER	00000120	G
ABORT_PKT	00001B39	R	03	CDB_G_HWA	00000250	G
ACBSL_KAST	= 00000018			CDB_G_MAPPED	00000162	G
ACCESS	00000DDD	R	03	CDB_G_MLTTBL	000002A6	G
ADDRCVLIST	00001435	RG	03	CDB_G_MULTI	0000025E	G
ADD_MULTI	00002493	R	03	CDB_G_PHA	0000024A	G
ALLOC_CDB	00001EB3	R	03	CDB_G_PHYADR	00000256	G
ALLOC_P2BUF	U0000DF6	R	03	CDB_G_RRING	00000162	G
ALT_START	00000751	RG	03	CDB_G_XRING	000001CE	G
ASSEM_PKTS	00001BA1	RG	03	CDB_L_BIDCTR	0000014A	G
ATS_UBA	= 00000001			CDB_L_BRCCTR	0000012E	G
BAD_PARAM_TBL	000001A6	R	03	CDB_L_BS1CTR	00000146	G
BLD_IRP	00002275	R	03	CDB_L_BSMCTR	00000142	G
BLD_STOP_IRP	0000223C	R	03	CDB_L_BSNCTR	0000014E	G
BLK_B_SPARE	0000000B			CDB_L_CSR	00000114	G
BLK_B_TYPE	0000000A			CDB_L_DBRCTR	00000122	G
BLK_C_HEADER	0000000C			CDB_L_DBSCTR	0000013A	G
BLK_L_LINK	00000000			CDB_L_DEVDEPEND	0000010C	G
BLK_T_DATA	0000000C			CDB_L_FPC	0000000C	G
BLK_W_SIZE	00000008			CDB_L_FQBL	00000004	G
BRDCST1	= FFFFFFFF			CDB_L_FQFL	00000000	G
BRDCST2	= 0000FFFF			CDB_L_FR3	00000010	G
BUGS_NOBUFCKT	*****	X	03	CDB_L_FR4	00000014	G
BUGS_UNSUPRTCPU	*****	X	03	CDB_L_LASTCSR	00000010	G
CANSC_CANCEL	= 00000000			CDB_L_MBLCTR	00000126	G
CANSC_DASSGN	= 00000001			CDB_L_MBSCTR	0000013E	G
CANCEL	00002315	RG	03	CDB_L_MBYCTR	00000132	G
CCBSL_UCB	= 00000000			CDB_L_MSCTR	00000152	G
CDB_B_AQUOTA	000002EE	G		CDB_L_PRMUSER	00000210	G
CDB_B_CON	00000249	G		CDB_L_RCVMAP	0000001C	G
CDB_B_DIAG1	00000119	G		CDB_L_RCV_PA	000000AC	G
CDB_B_FIPL	0000000B	G		CDB_L_RCV_VA	000000C4	G
CDB_B_LASTRCV	00000104	G		CDB_L_RINGMAP	0000011C	G
CDB_B_LASTXMT	00000105	G		CDB_L_RRINGPA	00000044	G
CDB_B_MLT	00000248	G		CDB_L_RRINGVA	0000007C	G
CDB_B_MLTTBL	0000025D	G		CDB_L_TQE	00000214	G
CDB_B_MQUOTA	000002EF	G		CDB_L_UCBO	00000110	G
CDB_B_MULTI	0000025C	G		CDB_L_UV1BUF	0000020C	G
CDB_B_NEXTRCV	00000019	G		CDB_L_XMTMAP	00000038	G

..S

LO

EM

EM

LO

XQDRIVER
Symbol table

- VAX/VMS QNA driver

H 5

8-JAN-1985 17:49:06 VAX/VMS Macro V04-00 Page 180
5-SEP-1984 00:20:54 [DRIVER.BUGSRC]XQDRIVER.MAR;1 (81)

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_SRC	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\$C_INTD	= 00000024		
CSR	0000000E	G	
CXBSB_CODE	= 0000000B		
CXBSB_R_FLAGS	0000000B	G	
CXBSB_TYPE	= 0000000A		
CXBSB_XQ_FUNC	00000020	G	
CXBSB_XQ_RING	00000023	G	
CXBSB_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\$C_BC	= 00000004		
CXB\$C_END_ACTION	= 0000001C		
CXB\$C_FL	= 00000000		
CXB\$C_IRP	= 00000014		
CXB\$C_LINK	= 00000010		
CXB\$C_SPARE0	= 00000024		
CXB\$C_SPARE1	= 00000020		
CXB\$C_T_IRP	00000024	G	
CXB\$C_T_UCB	00000024	G	
CXB\$C_STATION	= 00000028		
CXB\$T_R_DATA	00000038	G	
CXB\$T_R_USERDAT	00000046	G	
CXB\$T_T_DATA	0000003A	G	
CXBSW_BCNT	= 0000001A		
CXBSW_BOFF	= 00000018		
CXBSW_LENGTH	= 0000000C		
CXBSW_R_NCHAIN	0000001C	G	
CXBSW_R_PTYPE	00000044	G	
CXBSW_R_SIZE	00000046	G	
CXBSW_R_STS	00000014	G	
CXBSW_SIZE	= 00000008		
CXBSW_XQ RID	00000022	G	
CXB_CHECKER	000023E4	R	03
CXB_CHECKPKT	00002425	R	03
DC\$SCOM	= 00000020		
DDBSL_DDT	= 0000000C		
DDBSL_UCB	= 00000004		
DELETE_BLOCK	0000226C	R	03
DELETE_SHR	000022AD	RG	03
DEVSM_AVL	= 00040000		
DEVSM_IDV	= 04000000		
DEVSM_NET	= 00002000		
DEVSM_ODV	= 08000000		
DEVSM_SHR	= 00010000		
DEV_TIMEOUT	00001E8B	RG	03

-\$
--
Ps
--
\$\$
HA
PC
RE
-V
-V
-V

XQDRIVER
Symbol table

- VAX/VMS QNA driver

I 5

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		
DPTSB_FLAGS	= 0000000D		
DPTSC_LENGTH	= 00000038		
DPTSC_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		
DT\$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		
EXESABORTIO	*****	X	03
EXESALLOCBUF	*****	X	03
EXESALONONPAGED	*****	X	03
EXESALOPHYCNTG	*****	X	03
EXESBUFRQUOTA	*****	X	03
EXESBUFRQUOPRC	*****	X	03
EXESDEANONPAGED	*****	X	03
EXESFINISHIO	*****	X	03
EXESFORK	*****	X	03
EXESGB_CPUTYPE	*****	X	03
EXESGL_TENUSEC	*****	X	03
EXESGL_UBDELAY	*****	X	03
EXESGO_SYSTIME	*****	X	03
EXESINSTIMG	*****	X	03
EXESPROBER_DSC	*****	X	03
EXESQIORETURN	*****	X	03
EXESREADCHK	*****	X	03

EXESWRITECHK	*****	X	03
FFISL_DL_UCB	= 00000034		
FFISL_ERROR	= 0000001C		
FFISL_RECV_DONE	= 00000018		
FFISL_SHUT_DONE	= 00000020		
FFISL_XMIT	= 00000010		
FFISL_XMIT_DONE	= 00000014		
FFI_INIT	0000026A	RG	03
FILCRCLIST	00001433	RG	03
FIND_MLNTENTRY	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
FKBSB_FIPL	= 0000000B		
FKBSB_TYPE	= 0000000A		
FKBSL_FR3	= 00000010		
FORK_PROC	00001663	RG	03
FORK_TIMER	00001321	R	03
FUNCTAB_LEN	= 00000040		
GET_CHAR_BUF	00000D7D	R	03
IDBSB_VECTOR	= 0000000B		
IDBSL_CSR	= 00000000		
IDBSL_UCBLST	= 00000018		
INACT_ERROR	000006C4	R	03
INIT_C_AQUOTA	= 00000002		
INIT_C_BUFSIZE	= 00000080		
INIT_C_QUOTA	= 00002328		
INTERIT	0000163A	R	03
IOSV_ATTNAST	= 00000008		
IOSV_CLR_COUNT	= 0000000A		
IOSV_CTRL	= 00000009		
IOSV_NOW	= 00000006		
IOSV_RD_COUNT	= 00000008		
IOSV_SHUTDOWN	= 00000007		
IOSV_STARTUP	= 00000006		
IOS_READBLK	= 00000021		
IOS_READPBLK	= 0000000C		
IOS_READVBLK	= 00000031		
IOS_SENSECHAR	= 0000001B		
IOS_SENSEMODE	= 00000027		
IOS_SETCHAR	= 0000001A		
IOS_SETMODE	= 0000C023		
IOS_VIRTUAL	= 0000003F		
IOS_WRITEBLK	= 00000020		
IOS_WRITEPBLK	= 0000000B		
IOS_WRITEVBLK	= 00000030		
IOCSALOUBAMAP	*****	X	03
IOCSCREDIT_UCB	*****	X	03
IOCSDELETE_UCB	*****	X	03
IOCSINITATE	*****	X	03
IOCSLOADUBAMAP	*****	X	03
IOCSLOADUBAMAPA	*****	X	03
IOCSLOADUBAMAPN	*****	X	03
IOCSMNTVER	*****	X	03
IOCSPURGDATAP	*****	X	03
IOCSRELDATAP	*****	X	03

_S
_Ps
_V
_V
_V
_V

XQDRIVER
Symbol table

- VAX/VMS QNA driver

B 6

8-JAN-1985 17:49:06 VAX/VMS Macro V04-00 Page 187
5-SEP-1984 00:20:54 [DRIVER.BUGSRC]XQDRIVER.MAR;1 (81)

```

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_ADDRVC       = 00000040
XQ_C_CNVSIZ       = 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 r.ame	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
\$\$\$!05_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
-----	-----
_\$255\$DUA18:[SHRLIB]NMALIBRY.MLB;1	1
-\$255\$DUA18:[SYS.OBJ]LIB.MLB;1	42
-\$255\$DUA18:[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)+EXECMLS/LIB+SHRLIB\$:NMALIBRY/LIB

