```
UUU          UUU  VVV         VVV        111       RRRRRRRRRRR        000000000     MMM         MMM
UUU          UUU  VVV         VVV        111       RRRRRRRRRRRR       000000000     MMM         MMM
UUU          UUU  VVV         VVV        111       RRRRRRRRRRRR       000000000     MMM         MMM
UUU          UUU  VVV         VVV     111111       RRR         RRR  000         000  MMMMMMM   MMMMMMM
UUU          UUU  VVV         VVV     111111       RRR         RRR  000         000  MMMMMMM   MMMMMMM
UUU          UUU  VVV         VVV     111111       RRR         RRR  000         000  MMMMMMM   MMMMMMM
UUU          UUU  VVV         VVV        111       RRR         RRR  000         000  MMM  MMM  MMM
UUU          UUU  VVV         VVV        111       RRR         RRR  000         000  MMM  MMM  MMM
UUU          UUU  VVV         VVV        111       RRR         RRR  000         000  MMM  MMM  MMM
UUU          UUU  VVV         VVV        111       RRRRRRRRRRRR     000         000  MMM       MMM
UUU          UUU  VVV         VVV        111       RRRRRRRRRRRR     000         000  MMM       MMM
UUU          UUU  VVV         VVV        111       RRRRRRRRRRRR     000         000  MMM       MMM
UUU          UUU  VVV         VVV        111       RRR   RRR        000         000  MMM       MMM
UUU          UUU  VVV         VVV        111       RRR   RRR        000         000  MMM       MMM
UUU          UUU  VVV         VVV        111       RRR   RRR        000         000  MMM       MMM
UUU          UUU     VVV   VVV           111       RRR      RRR     000         000  MMM       MMM
UUU          UUU     VVV   VVV           111       RRR      RRR     000         000  MMM       MMM
UUU          UUU     VVV   VVV           111       RRR      RRR     000         000  MMM       MMM
UUUUUUUUUUUUUUUU        VVV            111111111    RRR         RRR    000000000     MMM       MMM
UUUUUUUUUUUUUUUU        VVV            111111111    RRR         RRR    000000000     MMM       MMM
UUUUUUUUUUUUUUUU        VVV            111111111    RRR         RRR    000000000     MMM       MMM
UUUUUUUUUUUUUUUU        VVV            111111111    RRR         RRR    000000000     MMM       MMM
```

FILEID**XQBOOT

```
XX      XX    QQQQQQ    BBBBBBBB      000000      000000    TTTTTTTTTT
XX      XX    QQQQQQ    BBBBBBBB      000000      000000    TTTTTTTTTT
XX      XX  QQ      QQ  BB      BB  00      00  00      00      TT
XX      XX  QQ      QQ  BB      BB  00      00  00      00      TT
  XX  XX    QQ      QQ  BB      BB  00      00  00      00      TT
  XX  XX    QQ      QQ  BB      BB  00      00  00      00      TT
    XX      QQ      QQ  BBBBBBBB    00      00  00      00      TT
    XX      QQ      QQ  BBBBBBBB    00      00  00      00      TT
  XX  XX    QQ  QQ  QQ  BB      BB  00      00  00      00      TT
  XX  XX    QQ  QQ  QQ  BB      BB  00      00  00      00      TT
XX      XX  QQ    QQ    BB      BB  00      00  00      00      TT          ....
XX      XX  QQ    QQ    BB      BB  00      00  00      00      TT          ....
XX      XX    QQQQ  QQ  BBBBBBBB      000000      000000      TT          ....
XX      XX    QQQQ  QQ  BBBBBBBB      000000      000000      TT            ..


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

```
    1  0        module xqboot;
    2  0
    3  0        {*************************************************************************
    4  0        {*                                                                       *
    5  0        {*  Copyright (c) 1984                                                   *
    6  0        {*  by DIGITAL Equipment Corporation, Maynard, Mass.                     *
    7  0        {*                                                                       *
    8  0        {*  This software is furnished under a license and may be used and copied *
    9  0        {*  only in accordance with the terms of such license and with the       *
   10  0        {*  inclusion of the above copyright notice. This software or any other  *
   11  0        {*  copies thereof may not be provided or otherwise made available to any *
   12  0        {*  other person. No title to and ownership of the software is hereby     *
   13  0        {*  transferred.                                                         *
   14  0        {*                                                                       *
   15  0        {*  The information in this software is subject to change without notice *
   16  0        {*  and should not be construed as a commitment by DIGITAL Equipment     *
   17  0        {*  Corporation.                                                         *
   18  0        {*                                                                       *
   19  0        {*  DIGITAL assumes no responsibility for the use or reliability of its  *
   20  0        {*  software on equipment which is not supplied by DIGITAL.              *
   21  0        {*                                                                       *
   22  0        {*************************************************************************
   23  0        {
   24  0        {++
   25  0        {
   26  0        { FACILITY:
   27  0        {
   28  0        {       VAXELN and MicroVAX
   29  0        {
   30  0        { ABSTRACT:
   31  0        {
   32  0        {       This module contains a VAX QNA (QBUS/Ethernet) Controller
   33  0        {       bootstrap driver.
   34  0        {
   35  0        { AUTHOR:
   36  0        {
   37  0        {       Kris Barker 25-October-1983
   38  0        {
   39  0        { VERSION:
   40  0        {
   41  0        {       V1.0-00
   42  0        {
   43  0        { REVISED:
   44  0        {
   45  0        {       23-May-1984     Kris K. Barker      1) Changed polling routines to
   46  0        {       V1.0-01                                check for error/use bits in
   47  0        {                                              status word rather than flag
   48  0        {                                              word to fix extern. loopback
   49  0        {                                              and down-line load failures.
   50  0        {                                           2) Fixed zeroing of status word
   51  0        {                                              to fix 3rd LED shut-off problem.
   52  0        {                                           3) Added net stop routine to shut
   53  0        {                                              QNA off after down-line loading.
   54  0        {
   55  0        {--}
   56  0
```

```
    0
    0
    0       include
    0               $datalink;
    0
    0       const
    0
    0           { qna controller parameters }
    0
    0           qna_addr_size   = 6;                        {address size}
    0           qna_type_size   = 2;                        {protocol type size}
    0           qna_header_size = (2 * qna_addr_size) + qna_type_size;
    0           qna_crc_size    = 4;                        {CRC size}
    0
    0           xmt_list_length = 1;                {number of transmit ring descriptors}
    0           rec_list_length = 4;                {number of receive ring descriptors}
    0           setup_packet_size = %o200;
    0
    0           xq_devtype = 1;
    0
    0
    0           { Values for use bits in descriptor flag word }
    0
    0           _last_noerr = 0;            {last segment of message with no errors}
    0           _last_werr  = 1;            {last segment of message with errors}
    0           _not_using  = 2;            {qna is not yet using this descriptor}
    0           _using      = 3;            {qna using or used and not last segment}
    0
    0           { LED values (occur in bit positions 2-4) }
    0
    0           mode_base = 0;      {no multicasting, no promiscuous mode, timer=0}
    0           led_nop   = 0;      {no effect}
    0           led1_out  = 1*4;    {LED 1 off to indicate self test running}
    0           led2_out  = 2*4;    {LED 2 off to indicate internal loopback passed}
    0           led3_out  = 3*4;    {LED 3 off to indicate external loopback passed}
    0
    0           load_protocol_type = %x160; {DECnet MOP load protocol type}
    0
    0           max_wait_count     = 150000;  {max loop count for receive/transmit wait}
    0           delay_count        = 500;     {delay count for resetting interface}
    0
    0           xq_base_physical     = %o014440;
    0           qbus_base_physical = %x20000000;
    0
```

```
  101  0
  102  0
  103  0         type
  104  0
  105  0             byte = 0..255;
  106  0             word = 0..65535;
  107  0             signed = -32768..32767;
  108  0             nibble = 0..%xF;
  109  0
  110  0             datalink_message = datalink_data_message;
  111  0
  112  0             bit_array = [word] packed record
  113  0                 case boolean of
  114  0                     true  : ( bits : packed array [0..15] of boolean );
  115  0                     false : ( hex : packed array [0..3] of nibble );
  116  0                 end;
  117  0
  118  0             data_format(n:integer) = packed record
  119  0                                          length : word;
  120  0                                          data   : string(n)
  121  0                                      end; { record }
  122  0
  123  0             { A physical address }
  124  0             phy_address_high = 0..63;
  125  0             phy_address = [long] packed record
  126  0                 case integer of
  127  0                     0 : ( ptr  : ^anytype );
  128  0                     1 : ( full : integer );
  129  0                     2 : ( low  : word;
  130  0                           high : phy_address_high );
  131  0                     3 : ( fl   : word;
  132  0                           fh   : word )
  133  0                 end; { record }
  134  0
  135  0
  136  0             { QNA csr register }
  137  0
  138  0             csr_register = [word] packed record
  139  0                 rx_ena      : boolean;          {receiver enable}
  140  0                 rset        : boolean;          {reset}
  141  0                 nxm_int     : boolean;          {non-existant memory interrupt}
  142  0                 boot_diag   : boolean;          {boot diagnostic rom}
  143  0                 tx_invalid  : boolean;          {transmit list invalid}
  144  0                 rx_invalid  : boolean;          {receive list invalid}
  145  0                 inte        : boolean;          {interrupt enable}
  146  0                 txi         : boolean;          {transmit interrupt request}
  147  0                 iloop       : boolean;          {internal loopback - set is disabled}
  148  0                 eloop       : boolean;          {external loopback}
  149  0                 stimer_ena  : boolean;          {sanity timer enable}
  150  0                 transceiver : [pos(12)] boolean;{power is being supplied}
  151  0                 carrier     : boolean;          {sample of carrier sense signal}
  152  0                 rxi         : [pos(15)] boolean;{receive interrupt request}
  153  0                 end; { record }
  154  0
  155  0             { QNA register layout }
  156  0
  157  0             word_reg = [word] packed record
```

```
158  0              qbus_word : word;
159  0              end;
160  0
161  0          qna_registers = packed record
162  0              case boolean of
163  0                  true  : ( station_address   : array [1..6] of word_reg );
164  0                  false : ( rx_desc_list_low  : [pos(32),word] word;
165  0                            rx_desc_list_high : [word] word;
166  0                            tx_desc_list_low  : [word] word;
167  0                            tx_desc_list_high : [word] word;
168  0                            vector_address    : [word] word;
169  0                            csr               : csr_register );
170  0              end; { record }
171  0
172  0          register_ptr = ^qna_registers;
173  0
174  0
175  0          { Setup descriptor format }
176  0
177  0          one_line = packed record
178  0              addr_byte : [pos(8)] packed array [1..7] of byte;
179  0              end;
180  0
181  0          setup_half = [aligned(2)] packed record
182  0              addresses : packed array [1..6] of one_line;
183  0              zeros     : packed array [1..2] of one_line
184  0              end;
185  0
186  0          setup_descriptor = packed array [1..2] of setup_half;
187  0
188  0
189  0          { Descriptor format }
190  0
191  0          descriptor = packed record
192  0              use_status  : [pos(14)] _last_noerr.._using;    {flag word}
193  0              bufadr_high : phy_address_high;          {high order address}
194  0              hbo_beg     : boolean;
195  0              lbo_term    : boolean;
196  0
197  0              setup       : [pos(16+12)] boolean;
198  0              end_of_mess : boolean;                   {buffer is last in message}
199  0              chain_addr  : boolean;                   {address is chain address}
200  0              valid       : boolean;                   {valid bit}
201  0
202  0              bufadr_low  : word;                      {low order buffer address}
203  0              buffer_size : signed;                    {length of the message}
204  0              status      : status_words;              {two status words}
205  0              end; { record }
206  0
207  0          transmit_list_array = array [0..xmt_list_length] of descriptor;
208  0          receive_list_array  = array [0..rec_list_length] of descriptor;
209  0
210  0          status_words = [long] packed record
211  0              case integer of
212  0                  0 :     ( whole_thing : integer );
213  0
214  0                  1 : { status for transmit descriptors }
```

```
215  0                          ( count    : [pos(4)] 0..15; {collision count}
216  0                            fail     : boolean;        {no collision pulse}
217  0                            abort    : boolean;        {transmission aborted}
218  0                            ste16    : boolean;        {default sanity timer on}
219  0                            nocar    : boolean;        {no carrier}
220  0                            loss     : boolean;        {carrier loss during xmt}
221  0                            tx_use   : [pos(14)] _last_noerr..._using; {use status}
222  0                            tdr      : 0..%x3fff);     {time domain reflectometry}
223  0                    2 : { status for receive descriptors }
224  0                          ( ovf      : boolean;        {overflow}
225  0                            crcerr   : boolean;        {crc error}
226  0                            frame    : boolean;        {framing alignment error}
227  0                            short    : boolean;        {short packet}
228  0
229  0                            rbl_hi   : [pos(8)] 0..7;  {bits 8-10 of rx byte length}
230  0                            runt     : boolean;        {packed could not be flushed}
231  0                            discard  : boolean;        {packet discarded}
232  0                            esetup   : boolean;        {eloop/setup packet}
233  0                            rx_use   : [pos(14)] _last_noerr..._using;  {use status}
234  0                            rbl_low  : byte)           {high bits of byte length}
235  0                end; { record }
236  0
237  0            { Mode bits }
238  0
239  0            mode_flag = packed record
240  0                case boolean of
241  0                    true  : ( full : integer );
242  0                    false : ( multicast    : boolean;
243  0                              promiscuous  : boolean;   {promiscuous mode}
244  0                              led_value    : 0..3;      {LED value}
245  0                              sanity_timer : 0..7)      {sanity timer in 1/4 sec.}
246  0                end; { record }
247  0
248  0                qna_boot_structure = record
249  0                    transmit_list      : transmit_list_array;
250  0                    xmt_list_address   : phy_address;
251  0                    receive_list       : receive_list_array;
252  0                    rec_list_address   : phy_address;
253  0
254  0                    hardware_address    : datalink_address; {current system address}
255  0                    load_server_address : datalink_address; {address of load server}
256  0
257  0                    transmit_buffer    : datalink_message;
258  0                    xmt_buffer_address : phy_address;
259  0                    receive_buffer     : array [0..rec_list_length-1] of datalink_message;
260  0                    rec_buffer_address : array [0..rec_list_length-1] of phy_address;
261  0
262  0                    address_control_packet : setup_descriptor;      {setup packet}
263  0                    setup_packet_physical  : phy_address;   {address of setup packet}
264  0                    end;
265  0
266  0            qna_structure_pointer = ^qna_boot_structure;
267  0
268  0
```

```
  269  0
  270  0
  271  0        var
  272  0
  273  0            qna_ptr         : qna_structure_pointer;
  274  0            qna_reg         : register_ptr;             {qna registers pointer}
  275  0            receive_index   : integer;
  276  0
  277  0
```

```
  278  0
  279  0
  280  0        function net_init ( var devtype      : integer;
  281  1                            xptr            : ^anytype ) : boolean;
  282  1        {++
  283  1        {
  284  1        { This procedure is called to initialize the QNA controller
  285  1        {
  286  1        { Inputs:
  287  1        {
  288  1        {       xptr = pointer to some usable memory
  289  1        {
  290  1        { Outputs:
  291  1        {
  292  1        {       devtype = qna device type
  293  1        {
  294  1        {       controller initialized
  295  1        {
  296  1        {--}
  297  1        type
  298  1            station_addr = [word] packed record
  299  1                case boolean of
  300  1                    true  : ( full : word_reg );
  301  1                    false : ( address_byte : [byte] byte );
  302  1                end; { record }
  303  1
  304  1        var
  305  1            pa_byte : station_addr;
  306  1            i       : integer;
  307  1            xmt_index : integer;
  308  1            rec_index : integer;
  309  1            six_bytes : integer;
  310  1            status    : boolean;
  311  1
  312  1
  313  1        begin
                                        0006
                                        0006  NET_INIT:
                                C1FC    0006          .entry  NET_INIT,^m<dv,iv,r2,r3,r4,r5,r6,r7,r8>
                              5E 04 C2  0008          subl2   #4,sp
                     57 00000000 EF 9E  000B          movab   $DATA,r7

  314  1
  315  1
  316  1            { Save the location of the available memory }
  317  1
  318  1            qna_ptr := xptr;
                                        0012
                        08 A7 08 AC D0  0012          movl    08(ap),08(r7)

  319  1            qna_ptr^ := zero;
                                        0017
              08 B7 1ED8 8F 00 6E 00 2C 0017          movc5   #0,(sp),#0,#1ED8,@08(r7)

  320  1
  321  1            { Initialize the register and vector addresses }
  322  1
```

```
323  1          qna_reg :: integer := xg_base_physical + qbus_base_physical;
                                  0020
                04 A7 20001920 8F D0    0020          movl    #20001920,04(r7)

324  1
325  1          { Initialize the controller: toggle the reset bit and boot
326  1            diagnostic bit to reset the controller and turn on the LEDs }
327  1
328  1          write_register ( qna_reg^.csr, rset := true );
                                  0028
                       52 04 A7 D0    0028          movl    04(r7),r2
                       0E A2 02 B0    002C          movw    #2,0E(r2)

329  1          write_register ( qna_reg^.csr, rset := false );
                                  0030
                       52 04 A7 D0    0030          movl    04(r7),r2
                       0E A2 00 B0    0034          movw    #0,0E(r2)

330  1          delay;
                                  0038
                    065D CF 00 FB     0038          calls   #0,DELAY

331  1          write_register ( qna_reg^.csr, boot_diag := true );
                                  003D
                       52 04 A7 D0    003D          movl    04(r7),r2
                       0E A2 08 B0    0041          movw    #8,0E(r2)

332  1          delay;
                                  0045
                    0650 CF 00 FB     0045          calls   #0,DELAY

333  1          write_register ( qna_reg^.csr, boot_diag := false );
                                  004A
                       52 04 A7 D0    004A          movl    04(r7),r2
                       0E A2 00 B0    004E          movw    #0,0E(r2)

334  1          receive_index := 0;
                                  0052
                             67 D4     0052          clrl    (r7)

335  1
336  1          { Get the descriptor lists, buffers, and setup packet addresses }
337  1
338  1          with qna_ptr^ do
                                  0054
                       56 08 A7 D0    0054          movl    08(r7),r6

339  1              begin
                                  0058

340  2              xmt_list_address.ptr      := address ( transmit_list );
                                  0058
                          18 A6 66 DE    0058          moval   (r6),18(r6)

341  2              rec_list_address.ptr      := address ( receive_list );
                                  005C
                       58 A6 1C A6 DE    005C          moval   1C(r6),58(r6)
```

```
342  2              xmt_buffer_address.ptr     := address ( transmit_buffer );
                                    0061
                  0660 C6 68 A6 9E  0061          movab   68(r6),0660(r6)

343  2              setup_packet_physical.ptr := address(address_control_packet);
                                    0067
                1ED4 C6 1E54 C6 DE  0067          moval   1E54(r6),1ED4(r6)

344  2
345  2              { Initialize the setup frame }
346  2
347  2              for six_bytes := 1 to 6 do
                                    006E
                       54 01 D0     006E          movl    #1,r4
                                    0071  vcg.1:

348  2                  begin
                                    0071

349  3                  pa_byte.full := read_register (qna_reg^.station_address[six_bytes] );
                                    0071
                     53 54 54 C1    0071          addl3   r4,r4,r3
                     52 04 A7 D0    0075          movl    04(r7),r2
                  52 FE A243 9E     0079          movab   -02(r2)[r3],r2
                        50 62 B0    007E          movw    (r2),r0
                     FE AD 50 B0    0081          movw    r0,-02(fp)

350  3                  hardware_address::datalink_address_byte[six_bytes] :=
                                    0085
                5B A644 FE AD 90    0085          movb    -02(fp),5B(r6)[r4]

351  3                                            pa_byte.address_byte;
352  3                  end;
                                    008B
                     E2 54 06 F3    008B          aobleq  #6,r4,vcg.1

353  2
354  2               { Put the addresses into the setup descriptor }
355  2
356  2               update_multicast_addresses;
                                    008F
                      0558 CF 00 FB 008F          calls   #0,UPDATE_MULTICAST_ADDRESSES

357  2
358  2              { Set up the transmit descriptor list }
359  2               for i := 0 to xmt_list_length do
                                    0094
                           53 D4    0094          clrl    r3
                                    0096  vcg.2:

360  2                  with transmit_list[i] do
                                    0096
                     52 53 0C C5    0096          mull3   #C,r3,r2
                      52 6642 9E    009A          movab   (r6)[r2],r2

361  3                      begin
```

```
                                    009E

362  4                      use_status     := _not_using;
                                    009E
                    62 02 0E 02 F0  009E            insv    #2,#E,#2,(r2)

363  4                      chain_addr     := false;
                                    00A3

364  4                      valid          := false;
                                    00A3
                    03 A2 C0 8F 8A  00A3            bicb2   #C0,03(r2)

365  4                  end;
                                    00A8
                       EA 53 01 F3  00A8            aobleq  #1,r3,vcg.2

366  2
367  2
368  2                  { And the receive descriptor list }
369  2
370  2                  for rec_index := 0 to rec_list_length-1 do
                                    00AC
                             54 D4  00AC            clrl    r4
                                    00AE  vcg.3:

371  2                      begin
                                    00AE

372  3                          rec_buffer_address[rec_index].ptr :=
                                    00AE
                  52 54 000005F8 8F C5  00AE            mull3   #5F8,r4,r2
                1E44 C644 0664 C642 9E  00B6            movab   0664(r6)[r2],1E44(r6)[r4]

373  3                                              address ( receive_buffer[rec_index] );
374  3                          with receive_list[rec_index] do
                                    00BF
                          52 54 0C C5   00BF            mull3   #C,r4,r2
                       53 1C A642 9E    00C3            movab   1C(r6)[r2],r3

375  3                              begin
                                    00C8

376  4                              use_status  := _not_using;
                                    00C8
                       63 02 0E 02 F0   00C8            insv    #2,#E,#2,(r3)

377  4                              bufadr_high := rec_buffer_address[rec_index].high;
                                    00CD
                          52 54 02 78   00CD            ashl    #2,r4,r2
                    52 1E46 C642 06 00 EF  00D1         extzv   #0,#6,1E46(r6)[r2],r2
                        02 A3 06 00 52 F0  00D9         insv    r2,#0,#6,02(r3)

378  4                              bufadr_low  := rec_buffer_address[rec_index].low;
                                    00DF
                          52 54 54 C1   00DF            addl3   r4,r4,r2
                    04 A3 1E44 C642 B0  00E3            movw    1E44(r6)[r2],04(r3)
```

```
379  4                    buffer_size := -((size(datalink_message)+1) div 2);
                                  00EA
                  06 A3 FD04 8F B0    00EA          movw    #-2FC,06(r3)

380  4                    chain_addr  := false;
                                  00F0
                    03 A3 40 8F 8A    00F0          bicb2   #40,03(r3)

381  4                    valid       := true;
                                  00F5
                    03 A3 80 8F 88    00F5          bisb2   #80,03(r3)

382  4                    status      := zero;
                                  00FA
                       08 A3 D4       00FA          clrl    08(r3)

383  4                    end;
                                  00FD

384  3                end;
                                  00FD
                      AD 54 03 F3     00FD          aobleq  #3,r4,vcg.3

385  2
386  2                { Last entry is to chain }
387  2
388  2                with receive_list[rec_list_length] do
                                  010T

389  2                    begin
                                  0101

390  3                    chain_addr := true;
                                  0101
                       4F A6 40 8F 88     0101          bisb2   #40,4F(r6)

391  3                    use_status := _not_using;
                                  0106
                   4D A6 02 06 02 F0      0106          insv    #2,#6,#2,4D(r6)

392  3                    valid      := true;
                                  010C
                       4F A6 80 8F 88     010C          bisb2   #80,4F(r6)

393  3                    bufadr_high := rec_list_address.high;
                                  0111
                   52 5A A6 06 00 EF      0111          extzv   #0,#6,5A(r6),r2
                   4E A6 06 00 52 F0      0117          insv    r2,#0,#6,4E(r6)

394  3                    bufadr_low  := rec_list_address.low;
                                  011D
                       50 A6 58 A6 B0     011D          movw    58(r6),50(r6)

395  3                    end;
                                  0122

396  2
```

```
  397  2              { Pass the descriptor address to the controller }
  398  2              write_register ( qna_reg^.rx_desc_list_low, rec_list_address.low );
                                  0122
                          52 04 A7 D0     0122          movl    04(r7),r2
                       04 A2 58 A6 B0     0126          movw    58(r6),04(r2)

  399  2              write_register ( qna_reg^.rx_desc_list_high, rec_list_address.high );
                                  0128
                    53 5A A6 06 00 EF     0128          extzv   #0,#6,5A(r6),r3
                          52 04 A7 D0     0131          movl    04(r7),r2
                          06 A2 53 B0     0135          movw    r3,06(r2)

  400  2
  401  2
  402  2              { Set the load server address to the load server multicast address }
  403  2
  404  2              load_server_address := ''(171,00,00,01,00,00);
                                  0139
                   62 A6 FEC2 CF 06 28    0139          movc3   #6,$CODE,62(r6)

  405  2
  406  2              end; { with qna_ptr^ }
                                  0140

  407  1
  408  1            { Start the good citizen tests }
  409  1
  410  1            net_init := false;
                                  0140
                             58 94    0140          clrb    r8

  411  1
  412  1            { LED #1 out indicates start of test }
  413  1
  414  1            status := set_mode ( led1_out );
                                  0142
                             04 DD    0142          pushl   #4
                          04DD CF 01 FB    0144          calls   #1,SET_MODE

  415  1
  416  1            { Set up the transmit buffer for the tests }
  417  1
  418  1            with qna_ptr^.transmit_buffer do
                                  0149
                             52 08 A7 D0    0149          movl    08(r7),r2
                                56 52 D0    014D          movl    r2,r6

  419  1              begin
                                  0150

  420  2              dest_address := qna_ptr^.hardware_address;
                                  0150
                      68 A6 5C A2 06 28    0150          movc3   #6,5C(r2),68(r6)

  421  2              source_address := zero;
                                  0156
                   6E A6 06 00 6E 00 2C    0156          movc5   #0,(sp),#0,#6,6E(r6)
```

```
422  2              protocol_type  := %x6000;
                                    015D
                  74 A6 6000 8F B0  015D          movw    #6000,74(r6)

423  2              for i := 1 to datalink_max_packet_size do
                                    0163
                        53 01 D0  0163          movl    #1,r3
                                    0166  vcg.4:

424  2                  data[i] := i mod 512;
                                    0166
                        52 53 09 00 EF  0166          extzv   #0,#9,r3,r2
                         75 A643 52 90  016B          movb    r2,75(r6)[r3]
                    EE 53 000005EA 8F F3  0170          aobleq  #5EA,r3,vcg.4

425  2              end;
                                    0178

426  1
427  1            { Run the internal loopback test }
428  1
429  1            status := loopback_test ( true );
                                    0178
                           01 DD    0178          pushl   #1
                        01C6 CF 01 FB  017A          calls   #1,LOOPBACK_TEST

430  1
431  1            { If ok, then turn out the 2nd LED and run the external test }
432  1
433  1            if status then
                                    017F
                              29 50 E9  017F          blbc    r0,vcg.5

434  1                begin
                                    0182

435  2                status := set_mode ( led2_out );
                                    0182
                              08 DD  0182          pushl   #8
                         049D CF 01 FB  0184          calls   #1,SET_MODE
                              52 50 90  0189          movb    r0,r2

436  2                status := loopback_test ( false );
                                    018C
                              00 DD  018C          pushl   #0
                         01B2 CF 01 FB  018E          calls   #1,LOOPBACK_TEST
                              52 50 90  0193          movb    r0,r2

437  2
438  2                { If ok, turn out LED #3 and set the return status to true }
439  2
440  2                if status then
                                    0196
                              12 52 E9  0196          blbc    r2,vcg.5

441  2                    begin
```

```
                                    0199
442  3                 status := set_mode ( led3_out );
                                    0199
                              0C DD 0199          pushl   #C
                        0486  CF 01 FB 019B          calls   #1,SET_MODE
443  3                 reset_for_normal_operations;
                                    01A0
                           1D AF 00 FB 01A0          calls   #0,RESET_FOR_NORMAL_OPERATIONS
444  3                 devtype := xq_devtype;
                                    01A4
                           04 BC 01 D0 01A4          movl    #1,@04(ap)
445  3                 net_init := true;
                                    01A8
                             58 01 90 01A8          movb    #1,r8
446  3                 end
                                    01AB
                                    01AB  vcg.5:
447  2
448  2               end;
449  1
450  1       end;
                                    01AB
                             50 58 90 01AB          movb    r8,r0
                                   04 01AE          ret
                                    01AF
451  0
```

```
452  0
453  0
454  0      procedure net_stop;
455  1      {++
456  1      {
457  1      { This procedure is called following the down-line load to turn off the
458  1      { QNA
459  1      {
460  1      {--}
461  1
462  1      begin
                                01AF
                                01AF  NET_STOP:
                          C004  01AF            .entry  NET_STOP,^m<dv,iv,r2>
           5C 00000000 EF 9E    01B1            movab   $DATA,ap

463  1
464  1          { Simply set the reset bit }
465  1
466  1          write_register ( qna_reg^.csr, rset := true );
                                01B8
                 52 04 AC D0    01B8            movl    04(ap),r2
                 0E A2 02 B0    01BC            movw    #2,0E(r2)

467  1
468  1      end;
                                01C0
                          04    01C0            ret
                                01C1

469  0
```

```
  470  0
  471  0
  472  0        procedure reset_for_normal_operations;
  473  1        {++
  474  1        {
  475  1        { This procedure resets the device either following the initialization tests
  476  1        { or after a receive or transmit timeout.
  477  1        {
  478  1        {--}
  479  1
  480  1        begin
                                    01C1
                                    01C1  RESET_FOR_NORMAL_OPERATIONS:
                           C004     01C1          .entry  RESET_FOR_NORMAL_OPERATIONS,^m<dv,iv,r2>
                 52 00000000 EF 9E  01C3          movab   $DATA,r2

  481  1
  482  1            { Reset the device }
  483  1
  484  1            write_register ( qna_reg^.csr, rset := true );
                                    01CA
                       5C 04 A2 D0  01CA          movl    04(r2),ap
                       0E AC 02 B0  01CE          movw    #2,0E(ap)

  485  1            delay;
                                    01D2
                            04C3 CF 00 FB  01D2    calls   #0,DELAY

  486  1            write_register ( qna_reg^.csr, rset   := false );
                                    01D7
                       5C 04 A2 D0  01D7          movl    04(r2),ap
                       0E AC 00 B0  01DB          movw    #0,0E(ap)

  487  1            delay;
                                    01DF
                            04B6 CF 00 FB  01DF    calls   #0,DELAY

  488  1            write_register ( qna_reg^.csr, rx_ena := true,
                                    01E4
                       5C 04 A2 D0  01E4          movl    04(r2),ap
                     0E AC 0101 8F B0  01E8       movw    #101,0E(ap)

  489  1                                                 iloop  := true,
  490  1                                                 eloop  := false );
  491  1
  492  1            { Pass the descriptor address to the controller }
  493  1
  494  1            receive_index := 0;
                                    01EE
                             62 D4  01EE          clrl    (r2)

  495  1            write_register (qna_reg^.rx_desc_list_low,qna_ptr^.rec_list_address.low);
                                    01F0
                       50 08 A2 D0  01F0          movl    08(r2),r0
                       5C 04 A2 D0  01F4          movl    04(r2),ap
                    04 AC 58 A0 B0  01F8          movw    58[r0],04(ap)

  496  1            write_register (qna_reg^.rx_desc_list_high,qna_ptr^.rec_list_address.high);
```

```
                                        01FD
                          5C 08 A2 D0   01FD          movl    08(r2),ap
                    50 5A AC 06 00 EF   0201          extzv   #0,#6,5A(ap),r0
                          5C 04 A2 D0   0207          movl    04(r2),ap
                          06 AC 50 B0   020B          movw    r0,06(ap)

 497  1
 498  1        end;
                                        020F
                                04      020F          ret
                                        0210

 499  0
```

```
  500  0
  501  0
  502  0       procedure load_transmit_descriptor ( msize : integer;
  503  1                                            buffer_address : phy_address;
  504  1                                            setup_packet   : boolean );
  505  1       {++
  506  1       {
  507  1       { This function will load the transmit descriptor
  508  1       {
  509  1       { Inputs:
  510  1       {
  511  1       {       msize specifies the size of the message
  512  1       {       buffer_address is the physical address of the buffer
  513  1       {       setup_packet is true if the buffer is a setup packet
  514  1       {
  515  1       { Outputs:
  516  1       {
  517  1       {       Transmit descriptor initialized
  518  1       {
  519  1       {--}
  520  1       begin
                                    0210
                                    0210  LOAD_TRANSMIT_DESCRIPTOR:
                           C01C     0210          .entry  LOAD_TRANSMIT_DESCRIPTOR,^m<dv,iv,r2,r3,r4>
                        5E 0C C2    0212          subl2   #C,sp
                     52 00000000 EF 9E  0215      movab   $DATA,r2
                           54 04 BC DE  021C      moval   @04(ap),r4
                        FC AD 08 BC D0  0220      movl    @08(ap),-04(fp)

  521  1
  522  1           with qna_ptr^, qna_ptr^.transmit_list[0] do
                                    0225
                           53 08 A2 D0  0225      movl    08(r2),r3

  523  1           begin
                                    0229

  524  2           use_status  := not_using;
  525  2           hbo_beg     := false;
                                    0229
                        63 02 0E 02 F0  0229      insv    #2,#E,#2,(r3)
                                    022E
                        02 A3 40 8F 8A  022E      bicb2   #40,02(r3)

  526  2           lbo_term    := odd(msize);
                                    0233
                        6E 54 01 00 EF  0233      extzv   #0,#1,r4,(sp)
                        63 01 17 6E F0  0238      insv    (sp),#17,#1,(r3)

  527  2           buffer_size := -((msize+1) div 2);
                                    023D
                              54 D6  023D         incl    r4
                           54 02 C6  023F         divl2   #2,r4
                           54 54 CE  0242         mnegl   r4,r4
                        06 A3 54 F7  0245         cvtlw   r4,06(r3)

  528  2           bufadr_high := buffer_address.high;
```

```
                                                0249
                    04 AE FE AD 06 00 EF        0249          extzv   #0,#6,-02(fp),04(sp)
                    02 A3 06 00 04 AE F0        0250          insv    04(sp),#0,#6,02(r3)

529  2                  bufadr_low  := buffer_address.low;
                                                0257
                       04 A3 FC AD B0           0257          movw    -04(fp),04(r3)

530  2                  end_of_mess := true;
                                                025C

531  2                  valid       := true;
                                                025C
                       03 A3 A0 8F 88           025C          bisb2   #A0,03(r3)

532  2                  setup       := setup_packet;
                                                0261
                          63 01 1C 0C AC F0     0261          insv    0C(ap),#1C,#1,(r3)

533  2                  end;
                                                0267

534  1
535  1          end;
                                                0267
                                      04        0267          ret
                                                0268

536  0
537  0
```

```
  538  0
  539  0
  540  0        function start_transmission : boolean;
  541  1        {++
  542  1        {
  543  1        { This function will write the transmit descriptor address to the qna
  544  1        { and wait for the qna to transmit the packet and receive the looped
  545  1        { response (called by the loopback routine and the set_mode routine).
  546  1        {
  547  1        { Inputs:
  548  1        {
  549  1        {       none
  550  1        {
  551  1        { Outputs:
  552  1        {
  553  1        {       function returns the value of the csr on return
  554  1        {
  555  1        {--}
  556  1        var
  557  1            time_count : integer;
  558  1
  559  1        begin
                                    0268
                                    0268  START_TRANSMISSION:
                           C01C     0268          .entry  START_TRANSMISSION,^m<dv,iv,r2,r3,r4>
                54 00000000 EF 9E   026A          movab   $DATA,r4

  560  1
  561  1            { Write the descriptor address }
  562  1        write_register (qna_reg^.tx_desc_list_low,qna_ptr^.xmt_list_address.low);
                                    0271
                      5C 08 A4 D0   0271          movl    08(r4),ap
                      52 04 A4 D0   0275          movl    04(r4),r2
                   08 A2 18 AC B0   0279          movw    18(ap),08(r2)

  563  1        write_register (qna_reg^.tx_desc_list_high,qna_ptr^.xmt_list_address.high);
                                    027E
                      52 08 A4 D0   027E          movl    08(r4),r2
                53 1A A2 06 00 EF   0282          extzv   #0,#6,1A(r2),r3
                      52 04 A4 D0   0288          movl    04(r4),r2
                      0A A2 53 B0   028C          movw    r3,0A(r2)

  564  1
  565  1            { Wait for completion }
  566  1
  567  1        time_count := 0;
                                    0290
                           53 D4    0290          clrl    r3

  568  1        repeat
                                    0292
                      52 64 03 C5   0292          mull3   #3,(r4),r2
                      5C 08 A4 D0   0296          movl    08(r4),ap
                                    029A  vcg.6:

  569  2            time_count := time_count+1;
```

```
                                    029A
                          53 D6     029A          incl    r3

570  2          until (qna_ptr^.receive_list[receive_index].status.whole_thing<>0) or
                                    029C
                   24 AC42 D5       029C          tstl    24(ap)[r2]
                           09 12    02A0          bneq    vcg.7
                000249F0 8F 53 D1   02A2          cmpl    r3,#249F0
                           EF 15    02A9          bleq    vcg.6
                                    02AB  vcg.7:

571  2                (time_count > max_wait_count);
572  1
573  1          start_transmission := time_count < max_wait_count;
                                    02AB
                           52 94    02AB          clrb    r2
                000249F0 8F 53 D1   02AD          cmpl    r3,#249F0
                           02 18    02B4          bgeq    vcg.8
                           52 96    02B6          incb    r2
                                    02B8  vcg.8:

574  1
575  1      end;
                                    02B8
                        50 52 90    02B8          movb    r2,r0
                              04    02BB          ret
                                    02BC

576  0
```

```
  577  0
  578  0
  579  0        function internal_locp : boolean;
  580  1        {++
  581  1        {
  582  1        { This function handles the special case of internal loopback
  583  1        {
  584  1        {--}
  585  1
  586  1        var
  587  1            time_count : integer;
  588  1            csr        : csr_register;
  589  1
  590  1        begin
                                    02BC
                                    02BC  INTERNAL_LOOP:
                          C00C      02BC          .entry  INTERNAL_LOOP,^m<dv,iv,r2,r3>
                       5E 04 C2     02BE          subl2   #4,sp
                    52 00000000 EF  02C1          movab   $DATA,r2

  591  1
  592  1            { Write the descriptor address }
  593  1
  594  1            write_register (qna_reg^.tx_desc_list_low,
                                    02C8
                          50 08 A2  02C8          movl    08(r2),r0
                          5C 04 A2  02CC          movl    04(r2),ap
                       08 AC 18 A0  02D0          movw    18(r0),08(ap)

  595  1                                qna_ptr^.xmt_list_address.low);
  596  1            write_register (qna_reg^.tx_desc_list_high,
                                    02D5
                          5C 08 A2  02D5          movl    08(r2),ap
                    53 1A AC 06 00  02D9          extzv   #0,#6,1A(ap),r3
                          5C 04 A2  02DF          movl    04(r2),ap
                          0A AC 53  02E3          movw    r3,0A(ap)

  597  1                                qna_ptr^.xmt_list_address.high);
  598  1
  599  1            { Wait for the transmit interrupt bit }
  600  1
  601  1            time_count := 0;
                                    02E7
                             53 D4  02E7          clrl    r3

  602  1            repeat
                                    02E9
                                    02E9  vcg.9:

  603  2                csr:= read_register ( qna_reg^.csr );
                                    02E9
                          5C 04 A2  02E9          movl    04(r2),ap
                          50 0E AC  02ED          movw    0E(ap),r0
                          FE AD 50  02F1          movw    r0,-02(fp)

  604  2                time_count := time_count+1;
```

```
                                    02F5
                          53 D6     02F5          incl    r3

605  2              nop;
                                    02F7
                     03B0 CF 00 FB  02F7          calls   #0,NOP

606  2          until (csr.txi) or (time_count > max_wait_count);
                                    02FC
                    09 FE AD 07 E0  02FC          bbs     #7,-02(fp),vcg.10
                000: 9F0 8F 53 D1   0301          cmpl    r3,#249F0
                          DF 15     0308          bleq    vcg.9
                                    030A  vcg.10:

607  1
608  1          { Clear the transmit bit and erable the receiver }
609  1
610  1          write_register ( qna_reg^.csr, txi    := true,
                                    030A
                       5C 04 A2 D0  030A          movl    04(r2),ap
                  0E AC 0081 8F B0  030E          movw    #81,0E(ap)

611  1                                          iloop  := false,
612  1                                          rx_ena := true );
613  1
614  1          { Now wait for the receive to complete }
615  1
616  1          time_count := 0;
                                    0314
                          53 D4     0314          clrl    r3

617  1          repeat
                                    0316
                                    0316  vcg.11:

618  2              time_count := time_count+1;
                                    0316
                          53 D6     0316          incl    r3

619  2              nop;
                                    0318
                     038F CF 00 FB  0318          calls   #0,NOP

620  2          until (qna_ptr^.receive_list[receive_index].status.whole_thing<>0) or
                                    031D
                       50 62 03 C5  031D          mull3   #3,(r2),r0
                       5C 08 A2 D0  0321          movl    08(r2),ap
                        24 AC40 D5  0325          tstl    24(ap)[r0]
                             09 12  0329          bneq    vcg.12
                 000249F0 8F 53 D1  032B          cmpl    r3,#249F0
                             E2 15  0332          bleq    vcg.11
                                    0334  vcg.12:

621  2                (time_count > max_wait_count);
622  1
623  1          internal_loop := time_count < max_wait_count;
```

```
                                        0334
                                5C 94   0334            clrb    ap
                    000249F0 8F 53 D1   0336            cmpl    r3,#249F0
                                02 18   033D            bgeq    vcg.13
                                5C 96   033F            incb    ap
                                        0341    vcg.13:

 624  1
 625  1      end;
                                        0341
                             50 5C 90   0341            movb    ap,r0
                                   04   0344            ret
                                        0345

 626  0
```

```
627  0
628  0
629  0       function loopback_test ( internal : boolean ) : boolean;
630  1       {++
631  1       {
632  1       { This function will perform the indicated kind of loopback test (either
633  1       { internal or external) and return the results of the test.
634  1       {
635  1       { Inputs:
636  1       {
637  1       {       internal is true for internal loopback test
638  1       {
639  1       { Outputs:
640  1       {
641  1       {       function returns true if test passed
642  1       {
643  1       {--}
644  1       var
645  1
646  1           status     : boolean;
647  1           time_count : integer;
648  1           msize      : integer;
649  1
650  1       begin
                                        0345
                                        0345  LOOPBACK_TEST:
                                  COFC  0345          .entry  LOOPBACK_TEST,^m<dv,iv,r2,r3,r4,r5,r6,r7>
                     56 00000000 EF 9E  0347          movab   $DATA,r6
                           57 04 BC DE  034E          moval   @04(ap),r7

651  1
652  1           { Set the proper value in the csr }
653  1
654  1           write_register ( qna_reg^.csr, iloop  := not internal,
                                        0352
                                 54 D4  0352          clrl    r4
                              52 57 01 8D  0354       xorb3   #1,r7,r2
                                 53 52 9A  0358       movzbl  r2,r3
                                 52 53 98  035B       cvtbl   r3,r2
                              54 01 08 52 F0  035E    insv    r2,#8,#1,r4
                                 52 53 98  0363       cvtbl   r3,r2
                              54 01 09 52 F0  0366    insv    r2,#9,#1,r4
                                 52 04 A6 D0  036B    movl    04(r6),r2
                                 0E A2 54 B0  036F    movw    r4,0E(r2)

655  1                                             eloop  := not internal,
656  1                                             rx_ena := false );
657  1
658  1
659  1           { Write the descriptor address to start the transmission }
660  1
661  1           qna_ptr^.receive_buffer[receive_index] := zero;
                                        0373
                           53 66 000005F8 8F C5  0373       mull3   #5F8,(r6),r3
                                 52 08 A6 D0  037B       movl    08(r6),r2
                0664 C243 05F8 8F 00 6F 00 2C  037F       movc5   #0,(sp),#0,#5F8,0664(r2)[r3]

662  1
```

```
663  1          if internal then
                                    038A
                         05 57 E9   038A          blbc    r7,vcg.14

664  1              msize := 6
                                    038D
                         55 06 D0   038D          movl    #6,r5
                            05 11   0390          brb     vcg.15
                                    0392  vcg.14:

665  2          else
666  1              msize := size(datalink_message);
                                    0392
                          55 05F8 8F 3C 0392          movzwl  #5F8,r5
                                    0397  vcg.15:

667  1
668  1          load_transmit_descriptor ( msize, qna_ptr^.xmt_buffer_address, false );
                                    0397
                               00 DD 0397          pushl   #0
                            52 08 A6 D0 0399          movl    08(r6),r2
                             0660 C2 9F 039D          pushab  0660(r2)
                               55 DD 03A1          pushl   r5
                             FE68 CF 03 FB 03A3          calls   #3,LOAD_TRANSMIT_DESCRIPTOR

669  1
670  1          if internal then
                                    03A8
                               0A 57 E9 03A8          blbc    r7,vcg.16

671  1              status := internal_loop
                                    03AB
                             FF0C CF 00 FB 03AB          calls   #0,INTERNAL_LOOP
                                52 50 90 03B0          movb    r0,r2
                                   08 11 03B3          brb     vcg.17
                                    03B5  vcg.16:

672  2          else
673  1              status := start_transmission;
                                    03B5
                             FEAE CF 00 FB 03B5          calls   #0,START_TRANSMISSION
                                52 50 90 03BA          movb    r0,r2
                                    03BD  vcg.17:

674  1
675  1          if status then
                                    03BD
                               48 52 E9 03BD          blbc    r2,vcg.20

676  1              begin
                                    03C0

677  2              qna_ptr^.receive_list[receive_index].use_status := _not_using;
                                    03C0
                  53 66 00000060 8F C5 03C0          mull3   #60,(r6),r3
                                53 06 C0 03C8          addl2   #6,r3
                                   52 08 A6 D0 03CB          movl    08(r6),r2
```

```
                    1D A2 02 53 02 F0    03CF           insv    #2,r3,#2,1D(r2)

678  2                 if qna_ptr^.receive_buffer[receive_index]::string(msize) =
                                          03D5
                  52 66 000005F8 8F C5    03D5           mull3   #5F8,(r6),r2
                         54 08 A6 D0      03DD           movl    08(r6),r4
                 68 A4 0664 C442 55 29    03E1           cmpc3   r5,0664(r4)[r2],68(r4)
                               05 12      03E9           bneq    vcg.18

679  2                                        qna_ptr^.transmit_buffer::string(msize) then
680  2                     loopback_test := true
                                          03EB
                            53 01 90      03EB           movb    #1,r3
                               02 11      03EE           brb     vcg.19
                                          03F0  vcg.18:

681  3                 else
682  2                     loopback_test := false;
                                          03F0
                                53 94     03F0           clrb    r3
                                          03F2  vcg.19:

683  2                 qna_ptr^.receive_list[receive_index].status := zero;
                                          03F2
                             52 66 0C C5  03F2           mull3   #C,(r6),r2
                           52 24 A442 9E  03F6           movab   24(r4)[r2],r2
                                   62 D4  03FB           clrl    (r2)

684  2                 receive_index := ( receive_index + 1 ) mod rec_list_length;
                                          03FD
                             52 66 01 C1  03FD           addl3   #1,(r6),r2
                          66 52 02 00 EF  0401           extzv   #0,#2,r2,(r6)

685  2                 end
                                          0406
                                   02 11  0406           brb     vcg.21
                                          0408  vcg.20:

686  1             else
687  1                 loopback_test := false;
                                          0408
                                   53 94  0408           clrb    r3
                                          040A  vcg.21:

688  1
689  1         end;
                                          040A
                                50 53 90  040A           movb    r3,r0
                                      04  040D           ret
                                          040E

690  0
```

```
  691  0
  692  0
  693  0       function net_receive ( var buff  : string(buff_size);
  694  1                              buff_size : word ) : boolean;
  695  1
  696  1       {++
  697  1       {
  698  1       { This function is called to receive a message
  699  1       {
  700  1       { Inputs:
  701  1       {
  702  1       {       buff = buffer to receive message
  703  1       {       buff_size = buffer size
  704  1       {
  705  1       { Outputs:
  706  1       {
  707  1       {       Message received into callers' buffer
  708  1       {
  709  1       {--}
  710  1
  711  1       var
  712  1           source_address : datalink_address;
  713  1           dest_address   : datalink_address;
  714  1           protocol_type  : datalink_protocol_type;
  715  1           xlen           : integer;
  716  1           done           : boolean;
  717  1           timecount      : integer;
  718  1           status         : boolean;
  719  1
  720  1       begin
                                    040E
                                    040E  NET_RECEIVE:
                            CFFC    040E          .entry  NET_RECEIVE,^m<dv,iv,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
                         5E 18 C2   0410          subl2   #18,sp
                57 00000000 EF 9E   0413          movab   $DATA,r7

  721  1
  722  1            done := false;
                                    041A
                            5B 94   041A          clrb    r11

  723  1            timecount := 0;
                                    041C
                            59 D4   041C          clrl    r9

  724  1
  725  1            while not done do
                                    041E
                         03 5B E9   041E          blbc    r11,gen.1
                          010D 31   0421          brw     vcg.30
                                    0424  gen.1:
                08 AE 04 BC DE      0424          moval   @04(ap),08(sp)
                04 AE 08 BC DE      0429          moval   @08(ap),04(sp)
                                    042E  vcg.22:

  726  1                begin
```

```
                                    042E

727  2              repeat
                                    042E

728  3                  timecount := timecount + 1;
                                    042E
                         59 D6      042E          incl    r9

729  3                  nop;
                                    0430
                     0277 CF 00 FB  0430          calls   #0,NOP

730  3              until (qna_ptr^.receive_list[receive_index].status.whole_thing<>0) or
                                    0435
                        53 67 03 C5 0435          mull3   #3,(r7),r3
                        52 08 A7 D0 0439          movl    08(r7),r2
                         24 A243 D5 043D          tstl    24(r2)[r3]
                              09 12 0441          bneq    vcg.23
                    000249F0 8F 59 D1 0443        cmpl    r9,#249F0
                              E2 15 044A          bleq    vcg.22
                                    044C  vcg.23:

731  3                        (timecount > max_wait_count);
732  2
733  2              status := timecount < max_wait_count;
                                    044C
                              52 94 044C          clrb    r2
                    000249F0 8F 59 D1 044E        cmpl    r9,#249F0
                              02 18 0455          bgeq    vcg.24
                              52 96 0457          incb    r2
                                    0459  vcg.24:
                           5A 52 90 0459          movb    r2,r10

734  2
735  2                { Clear the interrupt bits }
736  2                write_register ( qna_reg^.csr, rxi     := true,
                                    045C
                           52 04 A7 D0 045C          movl    04(r7),r2
                    0E A2 8105 8F B0 0460          movw    #8105,0E(r2)

737  2                                                nxm_int := true,
738  2                                                rx_ena  := true,
739  2                                                iloop   := true );
740  2
741  2                with qna_ptr^ do
                                    0466
                           58 08 A7 D0 0466          movl    08(r7),r8

742  2                  begin
                                    046A

743  3
744  3                  net_receive := false;
                                    046A
                                6E 94 046A          clrb    (sp)

745  3
```

```
  746  3                    if status and not receive_list[receive_index].status.discard then
                                    046C
                              71 5A E9    046C          blbc    r10,vcg.26
                              56 67 D0    046F          movl    (r7),r6
                  52 56 00000060 8F C5    0472          mull3   #60,r6,r2
                              52 04 C0    047A          addl2   #4,r2
                           5E 25 A8 52 E0    047D          bbs     r2,25(r8),vcg.26

  747  3                        begin
                                    0482

  748  4                        source_address := receive_buffer[receive_index].source_address;
                                    0482
                  52 56 000005F8 8F C5    0482          mull3   #5F8,r6,r2
                F4 AD 066A C842 06 28    048A          movc3   #6,066A(r8)[r2],-0C(fp)

  749  4                        protocol_type  := receive_buffer[receive_index].protocol_type;
                                    0492
                     56 000002FC 8F C4    0492          mull2   #2FC,r6
                        52 0670 C846 3C    0499          movzwl  0670(r8)[r6],r2

  750  4
  751  4                        if ( protocol_type = load_protocol_type ) and
                                    049F
                     00000160 8F 52 D1    049F          cmpl    r2,#160
                              53 12    04A6          bneq    vcg.28
                           52 62 A8 9A    04A8          movzbl  62(r8),r2
                              08 52 E8    04AC          blbs    r2,vcg.25
                     62 A8 F4 AD 06 29    04AF          cmpc3   #6,-0C(fp),62(r8)
                              44 12    04B5          bneq    vcg.28
                                    04B7  vcg.25:

  752  4                           ( odd(load_server_address::datalink_address_byte[1])
  753  4                                or (source_address = load_server_address) ) then
  754  4                            begin
                                    04B7

  755  5
  756  5                            { Extract length and message out of the data buffer }
  757  5
  758  5                            with receive_buffer[receive_index] do
                                    04B7
                     52 67 000005F8 8F C5    04B7          mull3   #5F8,(r7),r2
                           56 0664 C842 9E    04BF          movab   0664(r8)[r2],r6

  759  5                               begin
                                    04C5

  760  6                               xlen := data :: data_format(1).length;
                                    04C5
                           52 0E A6 3C    04C5          movzwl  0E(r6),r2

  761  6                               buff := data :: data_format(xlen).data;
                                    04C9
            08 BE 04 AE 20 10 A6 52 2C    04C9          movc5   r2,10(r6),#20,04(sp),@08(sp)

  762  6                               end; { with }
```

```
                                    04D2

763  5
764  5                      load_server_address := source_address;
                                    04D2
                62 A8 F4 AD 06 28   04D2          movc3   #6,-0C(fp),62(r8)

765  5                      done := true;
                                    04D8
                      5B 01 90      04D8          movb    #1,r11

766  5                      net_receive := true;
                                    04DB
                      6E 01 90      04DB          movb    #1,(sp)

767  5                      end;
                                    04DE

768  4                    end
                                    04DE
                       1B 11        04DE          brb     vcg.28
                                    04E0  vcg.26:

769  3                  else
770  3                      with receive_list[receive_index].status do
                                    04E0
                     52 67 0C C5    04E0          mull3   #C,(r7),r2
                  52 24 A842 9E     04E4          movab   24(r8)[r2],r2

771  4                      if not status or (crcerr and discard and not runt) then
                                    04E9
                        0C 5A E9    04E9          blbc    r10,vcg.27
                     0B 62 01 E1    04EC          bbc     #1,(r2),vcg.28
                     07 62 0C E1    04F0          bbc     #C,(r2),vcg.28
                     03 62 0B E0    04F4          bbs     #B,(r2),vcg.28
                                    04F8  vcg.27:

772  5                        done := true;
                                    04F8
                        5B 01 90    04F8          movb    #1,r11
                                    04FB  vcg.28:

773  3
774  3                  end; { with }
                                    04FB

775  2
776  2                { Now update the receive list index }
777  2
778  2                if status then
                                    04FB
                          2D 5A E9  04FB          blbc    r10,vcg.29

779  2                  begin
                                    04FE

780  3                  qna_ptr^.receive_list[receive_index].use_status := _not_using;
```

```
                                        04FE
                53 67 00000060 8F C5    04FE            mull3   #60,(r7),r3
                         53 06 C0       0506            addl2   #6,r3
                      52 08 A7 D0       0509            movl    08(r7),r2
                   1D A2 02 53 02 F0    050D            insv    #2,r3,#2,1D(r2)

781  3                qna_ptr^.receive_list[receive_index].status := zero;
                                        0513
                      53 67 0C C5       0513            mull3   #C,(r7),r3
                      52 08 A7 D0       0517            movl    08(r7),r2
                    52 24 A243 9E       051B            movab   24(r2)[r3],r2
                            62 D4       0520            clrl    (r2)

782  3                receive_index := (receive_index+1) mod rec_list_length;
                                        0522
                      52 67 01 C1       0522            addl3   #1,(r7),r2
                   67 52 02 00 EF       0526            extzv   #0,#2,r2,(r7)

783  3                end;
                                        052B
                                        052B  vcg.29:

784  2
785  2            end; { while }
                                        052B
                         03 5B E8       052B            blbs    r11,vcg.30
                          FEFD 31       052E            brw     vcg.22
                                        0531  vcg.30:

786  1
787  1
788  1        end;
                                        0531
                               50 6E 90 0531            movb    (sp),r0
                                     04 0534            ret
                                        0535

789  0
```

```
  790  0
  791  0
  792  0        function net_transmit ( var buff  : string(buff_size);
  793  1                                buff_size : word ) : boolean;
  794  1
  795  1        {++
  796  1        {
  797  1        { This function is called to transmit a message
  798  1        {
  799  1        { Inputs:
  800  1        {
  801  1        {       buff = buffer
  802  1        {       buff_size = buffer size
  803  1        {
  804  1        { Outputs:
  805  1        {
  806  1        {       buffer transmitted
  807  1        {--}
  808  1        var
  809  1            csr         : csr_register;
  810  1            msize       : integer;
  811  1            timecount   : integer;
  812  1
  813  1        begin
                                    0535
                                    0535  NET_TRANSMIT:
                            C1FC    0535          .entry  NET_TRANSMIT,^m<dv,iv,r2,r3,r4,r5,r6,r7,r8>
                         5E 04 C2   0537          subl2   #4,sp
                      57 00000000 EF 9E  053A     movab   $DATA,r7

  814  1
  815  1            with qna_ptr^ do
                                    0541
                            56 08 A7 D0  0541     movl    08(r7),r6
  816  1                begin
                                    0545

  817  2
  818  2                with transmit_buffer do
                                    0545

  819  2                    begin
                                    0545

  820  3                    dest_address   := load_server_address;
                                    0545
                      68 A6 62 A6 06 28  0545     movc3   #6,62(r6),68(r6)
  821  3                    source_address := hardware_address;
                                    054B
                      6E A6 5C A6 06 28  054B     movc3   #6,5C(r6),6E(r6)
  822  3                    protocol_type  := load_protocol_type;
                                    0551
                       74 A6 0160 8F B0  0551     movw    #160,74(r6)
  823  3                    data :: data_format(1).length := buff_size;
```

```
                                0557
                   58 08 BC DE  0557          moval   @08(ap),r8
                   76 A6 58 B0  055B          movw    r8,76(r6)

824  3              data :: data_format(buff_size).data := buff
                                055F
                   52 04 AC D0  055F          movl    04(ap),r2
                78 A6 62 58 28  0563          movc3   r8,(r2),78(r6)

825  3              end; { with }
826  2
827  2         { Load the transmit descriptor }
828  2
829  2         msize       := buff_size + datalink_header_size + 2;
                                0568
                      58 0E C0  0568          addl2   #E,r8
                   52 58 02 C1  056B          addl3   #2,r8,r2

830  2         if msize < datalink_min_packet_size then
                                056F
                      3C 52 D1  056F          cmpl    r2,#3C
                         03 18  0572          bgeq    vcg.31

831  2           msize := datalink_min_packet_size;
                                0574
                      52 3C D0  0574          movl    #3C,r2
                                0577  vcg.31:

832  2         load_transmit_descriptor ( msize, xmt_buffer_address, false );
                                0577
                            00 DD  0577       pushl   #0
                      0660 C6 9F   0579       pushab  0660(r6)
                            52 DD  057D       pushl   r2
                    FC8C CF 03 FB  057F       calls   #3,LOAD_TRANSMIT_DESCRIPTOR

833  2
834  2         { Write the descriptor address to start the transmission }
835  2
836  2         write_register ( qna_reg^.tx_desc_list_low, xmt_list_address.low );
                                0584
                      52 04 A7 D0  0584       movl    04(r7),r2
                   08 A2 18 A6 B0  0588       movw    18(r6),08(r2)

837  2         write_register ( qna_reg^.tx_desc_list_high, xmt_list_address.high );
                                058D
                    53 1A A6 06 00 EF  058D    extzv   #0,#6,1A(r6),r3
                      52 04 A7 D0  0593       movl    04(r7),r2
                      0A A2 53 B0  0597       movw    r3,0A(r2)

838  2
839  2         end; { with }
                                059B

840  1
841  1       { Wait for completion }
842  1
843  1       timecount := 0;
```

```
                                059B
                     53 D4      059B          clrl    r3

844  1          repeat
                                059D
                                059D  vcg.32:

845  2              csr := read_register ( qna_reg^.csr );
                                059D
                     52 04 A7 D0    059D          movl    04(r7),r2
                     50 0E A2 B0    05A1          movw    0E(r2),r0
                     FE AD 50 B0    05A5          movw    r0,-02(fp)

846  2              timecount := timecount + 1;
                                05A9
                     53 D6      05A9          incl    r3

847  2              nop;
                                05AB
                      00FC CF 00 FB 05AB          calls   #0,NOP

848  2          until csr.txi or ( timecount > max_wait_count );
                                05B0
                        09 FE AD 07 E0  05B0          bbs     #7,-02(fp),vcg.33
                    000249F0 8F 53 D1   05B5          cmpl    r3,#249F0
                             DF 15      05BC          bleq    vcg.32
                                05BE  vcg.33:

849  1          write_register ( qna_reg^.csr, txi := true,
                                05BE
                           52 04 A7 D0  05BE          movl    04(r7),r2
                     0E A2 0185 8F B0   05C2          movw    #185,0E(r2)

850  1                                              nxm_int := true,
851  1                                              rx_ena  := true,
852  1                                              iloop   := true );
853  1
854  1          { Set the return status.  If the transmit timed out, reset the device }
855  1
856  1          if ( csr.nxm_int ) or ( not csr.txi ) or
                                05C8
                        14 FE AD 02 E0  05C8          bbs     #2,-02(fp),vcg.34
                        0F FE AD 07 E1  05CD          bbc     #7,-02(fp),vcg.34
                           52 08 A7 D0  05D2          movl    08(r7),r2
                    52 09 A2 02 06 EF   05D6          extzv   #6,#2,09(r2),r2
                                01 52 D1  05DC          cmpl    r2,#1
                                   04 12  05DF          bneq    vcg.35
                                05E1  vcg.34:

857  1             ( qna_ptr^.transmit_list[0].status.tx_use = _last_werr ) then
858  1              net_transmit := false
                                05E1
                                52 94  05E1          clrb    r2
                                03 11  05E3          brb     vcg.36
                                05E5  vcg.35:

859  2          else
```

```
860  1                  net_transmit := true;
                                  05E5
                        52 01 90  05E5          movb    #1,r2
                                  05E8  vcg.36:
861  1
862  1          end;
                                  05E8
                        50 52 90  05E8          movb    r2,r0
                              04  05EB          ret
                                  05EC
863  0
```

```
  procedure update_multicast_addresses;
  {++
  {
  { This procedure takes the current saved multicast address list and uses
  { it to fill the setup descriptor.
  {
  { Inputs:
  {
  {       multicast address save list
  {
  { Outputs:
  {
  {       updated setup descriptor
  {
  { The address control packet looks like:
  {
  {                          0   1   2   3   4   5   6   7
  {                        +-------------------------------+
  {              0         | 0 | P | B | M | M | M | M | M |
  {                        +   + H + R + U + U + U + U + U +
  {             10         | 0 | Y | O | L | L | L | L | L |
  {                        +   + S + A + T + T + T + T + T +
  {             20         | 0 | I | D | I | I | I | I | I |
  {                        +   + C + C + C + C + C + C + C +
  {             30         | 0 | A | A | A | A | A | A | A |
  {                        +   + L + S + S + S + S + S + S +
  {             40         | 0 |   | T | T | T | T | T | T |
  {                        +   +   +   +   +   +   +   +   +
  {             50         | 0 |ADR|ADR| 1 | 2 | 3 | 4 | 5 |
  {                        |   +---+---+---+---+---+---+---+
  {             60         | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  {                        +   +   +   +   +   +   +   +   +
  {             70         | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  {                        +   +---+---+---+---+---+---+---+
  {            100         | 0 | M | M | M | M | M | M | M |
  {                        +   + U + U + U + U + U + U + U +
  {            110         | 0 | L | L | L | L | L | L | L |
  {                        +   + T + T + T + T + T + T + T +
  {            120         | 0 | I | I | I | I | I | I | I |
  {                        +   + C + C + C + C + C + C + C +
  {            130         | 0 | A | A | A | A | A | A | A |
  {                        +   + S + S + S + S + S + S + S +
  {            140         | 0 | T | T | T | T | T | T | T |
  {                        +   +   +   +   +   +   +   +   +
  {            150         | 0 | 6 | 7 | 8 | 9 |10 |11 |12 |
  {                        +   +---+---+---+---+---+---+---+
  {            160         | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  {                        +   +   +   +   +   +   +   +   +
  {            170         | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
  {                        +-------------------------------+
  {
  {--}
  var
      i,j,k     : integer;

  begin
```

```
                                        05EC
                                        05EC  UPDATE_MULTICAST_ADDRESSES:
                                  C1FC  05EC          .entry  UPDATE_MULTICAST_ADDRESSES,^m<dv,iv,r2,r3,r4,r5,r6,r7,r8>
                   57 00000000 EF 9E    05EE          movab   $DATA,r7

 921  1
 922  1          { Load all addresses as the hardware address }
 923  1
 924  1          with qna_ptr^ do
                                        05F5
                      52 08 A7 D0       05F5          movl    08(r7),r2

 925  1              for i := 1 to 2 do
                                        05F9
                         58 01 D0       05F9          movl    #1,r8
                                        05FC  vcg.37:

 926  2               for j := 1 to 6 do
                                        05FC
                         53 01 D0       05FC          movl    #1,r3
                      56 58 06 78       05FF          ashl    #6,r8,r6
                                        0603  vcg.38:

 927  3                 for k := 1 to 7 do
                                        0603
                         54 01 D0       0603          movl    #1,r4
                      55 53 03 78       0606          ashl    #3,r3,r5
                         55 56 C0       060A          addl2   r6,r5
                                        060D  vcg.39:

 928  4                      address_control_packet[i].addresses[j].addr_byte[k] :=
                                        060D
                               5C 55 54 C1      060D          addl3   r4,r5,ap
                   1E0C C24C 5B A243 90         0611          movb    5B(r2)[r3],1E0C(r2)[ap]
                             F0 54 07 F3        0619          aobleq  #7,r4,vcg.39
                             E2 53 06 F3        061D          aobleq  #6,r3,vcg.38
                             D7 58 02 F3        0621          aobleq  #2,r8,vcg.37

 929  5                                             hardware_address::datalink_address_byte[j];
 930  1
 931  1          end;
                                        0625
                                  04    0625          ret
                                        0626

 932  0
```

```
 933  0
 934  0
 935  0      function set_mode ( led_value_mode : integer ) : boolean;
 936  1      {++
 937  1      {
 938  1      { This procedure will write the mode to the qna
 939  1      {
 940  1      { Inputs  : Current mode flags are used to set the mode
 941  1      {
 942  1      { Outputs : Controller mode set
 943  1      {
 944  1      {--}
 945  1      var
 946  1          mode_multiplier : integer;
 947  1          tsize           : integer;
 948  1          time_count      : integer;
 949  1
 950  1      begin
                                    0626
                                    0626  SET_MODE:
                             C00C   0626          .entry  SET_MODE,^m<dv,iv,r2,r3>
                  52 00000000 EF 9E 0628          movab   $DATA,r2

 951  1
 952  1          { Load the transmit descriptor }
 953  1
 954  1          mode_multiplier := mode_base + led_value_mode;
                                    062F
                51 04 AC 00000080 8F C1 062F          addl3   #80,04(ap),r1

 955  1          load_transmit_descriptor ( (setup_packet_size+mode_multiplier),
                                    0638
                            01 DD   0638          pushl   #1
                         50 08 A2 D0 063A          movl    08(r2),r0
                          1ED4 C0 9F 063E          pushab  1ED4(r0)
                            51 DD   0642          pushl   r1
                       FBC7 CF 03 FB 0644          calls   #3,LOAD_TRANSMIT_DESCRIPTOR

 956  1                                       qna_ptr^.setup_packet_physical,
 957  1                                       true );
 958  1
 959  1          { Wait for completion }
 960  1
 961  1          set_mode := true;
                                    0649
                            53 01 90 0649          movb    #1,r3

 962  1          if start_transmission then
                                    064C
                       FC17 CF 00 FB 064C          calls   #0,START_TRANSMISSION
                            40 50 E9 0651          blbc    r0,vcg.40

 963  1              begin
                                    0654

 964  2              qna_ptr^.receive_list[receive_index].use_status := _not_using;
```

```
                                        0654
                51 62 00000060 8F C5    0654          mull3   #60,(r2),r1
                         51 06 C0       065C          addl2   #6,r1
                      50 08 A2 D0       065F          movl    08(r2),r0
                   1D A0 02 51 02 F0    0663          insv    #2,r1,#2,1D(r0)

965  2            qna_ptr^.receive_list[receive_index].status := zero;
                                        0669
                      51 62 0C C5       0669          mull3   #C,(r2),r1
                      50 08 A2 D0       066D          movl    08(r2),r0
                    50 24 A041 9E       0671          movab   24(r0)[r1],r0
                            60 D4       0676          clrl    (r0)

966  2            receive_index := ( receive_index + 1 ) mod rec_list_length;
                                        0678
                      50 62 01 C1       0678          addl3   #1,(r2),r0
                   62 50 02 00 EF       067C          extzv   #0,#2,r0,(r2)

967  2            if qna_ptr^.transmit_list[0].status.tx_use = _last_werr then
                                        0681
                      50 08 A2 D0       0681          movl    08(r2),r0
                50 09 A0 02 06 EF       0685          extzv   #6 #2,09(r0),r0
                         01 50 D1       068B          cmpl    r0,#1
                            06 12       068E          bneq    vcg.41

968  2                set_mode := false;
                                        0690
                            53 94       0690          clrb    r3

969  2                end
970  1            else
                                        0692
                         02 11          0692          brb     vcg.41
                                        0694  vcg.40:

971  1                set_mode := false;
                                        0694
                         53 94          0694          clrb    r3
                                        0696  vcg.41:

972  1
973  1        end;
                                        0696
                      50 53 90          0696          movb    r3,r0
                            04          0699          ret
                                        069A

974  0
```

```
  975  0
  976  0
  977  0        procedure delay;
  978  1
  979  1        var
  980  1
  981  1            i : integer;
  982  1
  983  1        begin
                                    069A
                                    069A  DELAY:
                              C000  069A            .entry  DELAY,^m<dv,iv>

  984  1
  985  1            for i := 1 to delay_count do
                                    069C
                         5C 01 D0   069C            movl    #1,ap
                                    069F  vcg.42:

  986  1                nop
                                    069F
                            09 AF   069F            calls   #0,P'OP
                  F4 5C 000001F4 8F F3  06A3        aobleq  #1F4,ap,vcg.42
                               04   06AB            ret
                                    06AC

  987  2
  988  2        end;
  989  0
  990  0
  991  0        procedure nop;
  992  1        begin
                                    06AC
                                    06AC  NOP:
                              C000  06AC            .entry  NOP,^m<dv,iv>

  993  1        end;
                                    06AE
                              04    06AE            ret
                                    06AF

  994  0
  995  0
  996  0        end; { module xqboot }
  997  0

EPASCAL/OPT/MACH/LIS=LIS$:/OBJ=OBJ$: VMB$:XQBOOT+ELN$:RTLOBJECT/LIB
```

0431 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

WELCOME
TXT

LOGIN
COM

SYSTARTUP
COM

XQBOOT
LIS

EDTINI
EDT

REMOVE
COM

VERIFY
LIS

VERIFY

UVMS

UVSTARTUP
COM

VERIFY
MAP

ADDUSER
COM

MODPARAMS
DAT

RESTUSER
COM

BACKUSER
COM

SUCCESS
TXT

SYLOGIN
COM