```
UUU         UUU  VVV         VVV        111        RRRRRRRRRRRR        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  MMMMMM   MMMMMM
UUU         UUU  VVV         VVV     111111        RRR         RRR  000         000  MMMMMM   MMMMMM
UUU         UUU  VVV         VVV     111111        RRR         RRR  000         000  MMMMMM   MMMMMM
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
UUUUUUUUUUUUUUU        VVV           111111111     RRR         RRR     000000000     MMM         MMM
UUUUUUUUUUUUUUU        VVV           111111111     RRR         RRR     000000000     MMM         MMM
UUUUUUUUUUUUUUU        VVV           111111111     RRR         RRR     000000000     MMM         MMM
```

B00
V03-

**FILE**ID**XQBOOT



```
XX      XX    QQQQQQ    BBBBBBBB      OOOOOO      OOOOOO    TTTTTTTTTT
XX      XX    QQQQQQ    BBBBBBBB      OOOOOO      OOOOOO    TTTTTTTTTT
XX      XX  QQ      QQ  BB      BB  OO      OO  OO      OO      TT
XX      XX  QQ      QQ  BB      BB  OO      OO  OO      OO      TT
  XX  XX    QQ      QQ  BB      BB  OO      OO  OO      OO      TT
  XX  XX    QQ      QQ  BB      BB  OO      OO  OO      OO      TT
    XX      QQ      QQ  BBBBBBBB    OO      OO  OO      OO      TT
    XX      QQ      QQ  BBBBBBBB    OO      OO  OO      OO      TT
  XX  XX    QQ  QQ  QQ  BB      BB  OO      OO  OO      OO      TT
  XX  XX    QQ  QQ  QQ  BB      BB  OO      OO  OO      OO      TT
XX      XX  QQ    QQ    BB      BB  OO      OO  OO      OO      TT          ....
XX      XX  QQ    QQ    BB      BB  OO      OO  OO      OO      TT          ....
XX      XX    QQQQ  QQ  BBBBBBBB      OOOOOO      OOOOOO        TT          ....
XX      XX    QQQQ  QQ  BBBBBBBB      OOOOOO      OOOOOO        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
```

```
module xqboot;

{**************************************************************************
{*                                                                        *
{*  Copyright (c) 1984                                                    *
{*  by DIGITAL Equipment Corporation, Maynard, Mass.                      *
{*                                                                        *
{*  This software is furnished under a license and may be used and  copied *
{*  only  in  accordance  with  the  terms  of  such  license and with the *
{*  inclusion of the above copyright notice.  This software or  any  other *
{*  copies  thereof may not be provided or otherwise made available to any *
{*  other person.  No title to and ownership of  the  software  is  hereby *
{*  transferred.                                                          *
{*                                                                        *
{*  The information in this software is subject to change  without  notice *
{*  and  should  not  be  construed  as  a commitment by DIGITAL Equipment *
{*  Corporation.                                                          *
{*                                                                        *
{*  DIGITAL assumes no responsibility for the use or  reliability  of  its *
{*  software on equipment which is not supplied by DIGITAL.               *
{*                                                                        *
{**************************************************************************
{
{++
{
{ FACILITY:
{
{       VAXELN and MicroVAX
{
{ ABSTRACT:
{
{       This module contains a VAX QNA (QBUS/Ethernet) Controller
{       bootstrap driver.
{
{ AUTHOR:
{
{       Kris Barker 25-October-1983
{
{ VERSION:
{
{       V1.0-00
{
{ REVISED:
{
{       23-May-1984     Kris K. Barker      1) Changed polling routines to
{       V1.0-01                                check for error/use bits in
{                                              status word rather than flag
{                                              word to fix extern. loopback
{                                              and down-line load failures.
{                                           2) Fixed zeroing of status word
{                                              to fix 3rd LED shut-off problem.
{                                           3) Added net_stop routine to shut
{                                              QNA off after down-line loading.
{
{--}
```

```

    include
          $datalink;

    const

        { qna controller parameters }

        qna_addr_size    = 6;                           {address size}
        qna_type_size    = 2;                           {protocol type size}
        qna_header_size = (2 * qna_addr_size) + qna_type_size;
        qna_crc_size     = 4;                           {CRC size}

        xmt_list_length = 1;                {number of transmit ring descriptors}
        rec_list_length = 4;                {number of receive ring descriptors}
        setup_packet_size = %o200;

        xq_devtype = 1;


        { Values for use bits in descriptor flag word }

        _last_noerr = 0;            {last segment of message with no errors}
        _last_werr  = 1;            {last segment of message with errors}
        _not_using  = 2;            {qna is not yet using this descriptor}
        _using      = 3;            {qna using or used and not last segment}

        { LED values (occur in bit positions 2-4) }

        mode_base = 0;        {no multicasting, no promiscuous mode, timer=0}
        led_nop   = 0;        {no effect}
        led1_out  = 1*4;      {LED 1 off to indicate self test running}
        led2_out  = 2*4;      {LED 2 off to indicate internal loopback passed}
        led3_out  = 3*4;      {LED 3 off to indicate external loopback passed}

        load_protocol_type = %x160; {DECnet MOP load protocol type}

        max_wait_count      = 150000;  {max loop count for receive/transmit wait}
        delay_count         = 500;     {delay count for resetting interface}

        xq_base_physical      = %o014440;
        qbus_base_physical = %x20000000;

```

```


type

     byte = 0..255;
     word = 0..65535;
     signed = -32768..32767;
     nibble = 0..%xF;

     datalink_message = datalink_data_message;

     bit_array = [word] packed record
         case boolean of
             true  : ( bits : packed array [0..15] of boolean );
             false : ( hex : packed array [0..3] of nibble );
         end;

     data_format(n:integer) = packed record
                                  length : word;
                                  data   : string(n)
                              end; { record }

     { A physical address }
     phy_address_high = 0..63;
     phy_address = [long] packed record
         case integer of
             0 : ( ptr  : ^anytype );
             1 : ( full : integer );
             2 : ( low  : word;
                   high : phy_address_high );
             3 : ( fl   : word;
                   fh   : word )
         end; { record }


     { QNA csr register }

     csr_register = [word] packed record
         rx_ena      : boolean;           {receiver enable}
         rset        : boolean;           {reset}
         nxm_int     : boolean;           {non-existant memory interrupt}
         boot_diag   : boolean;           {boot diagnostic rom}
         tx_invalid  : boolean;           {transmit list invalid}
         rx_invalid  : boolean;           {receive list invalid}
         inte        : boolean;           {interrupt enable}
         txi         : boolean;           {transmit interrupt request}
         iloop       : boolean;           {internal loopback - set is disabled}
         eloop       : boolean;           {external loopback}
         stimer_ena  : boolean;           {sanity timer enable}
         transceiver : [pos(12)] boolean; {power is being supplied}
         carrier     : boolean;           {sample of carrier sense signal}
         rxi         : [pos(15)] boolean; {receive interrupt request}
         end; { record }

     { QNA register layout }

     word_reg = [word] packed record
```

```
          qbus_word : word;
          end;

     qna_registers = packed record
          case boolean of
               true  : ( station_address   : array [1..6] of word_reg );
               false : ( rx_desc_list_low  : [pos(32),word] word;
                         rx_desc_list_high : [word] word;
                         tx_desc_list_low  : [word] word;
                         tx_desc_list_high : [word] word;
                         vector_address    : [word] word;
                         csr               : csr_register );
          end; { record }

     register_ptr = ^qna_registers;


     { Setup descriptor format }

     one_line = packed record
          addr_byte : [pos(8)] packed array [1..7] of byte;
          end;

     setup_half = [aligned(2)] packed record
          addresses : packed array [1..6] of one_line;
          zeros     : packed array [1..2] of one_line
          end;

     setup_descriptor = packed array [1..2] of setup_half;


     { Descriptor format }

     descriptor = packed record
          use_status  : [pos(14)] _last_noerr.._using;    {flag word}
          bufadr_high : phy_address_high;             {high order address}
          hbo_beg     : boolean;
          lbo_term    : boolean;

          setup       : [pos(16+12)] boolean;
          end_of_mess : boolean;                      {buffer is last in message}
          chain_addr  : boolean;                      {address is chain address}
          valid       : boolean;                      {valid bit}

          bufadr_low  : word;                         {low order buffer address}
          buffer_size : signed;                       {length of the message}
          status      : status_words;                 {two status words}
          end; { record }

     transmit_list_array = array [0..xmt_list_length] of descriptor;
     receive_list_array  = array [0..rec_list_length] of descriptor;

     status_words = [long] packed record
          case integer of
               0 :     ( whole_thing : integer );

               1 : { status for transmit descriptors }
```

```
                        ( count    : [pos(4)] 0..15; {collision count}
                          fail     : boolean;        {no collision pulse}
                          abort    : boolean;        {transmission aborted}
                          ste16    : boolean;        {default sanity timer on}
                          nocar    : boolean;        {no carrier}
                          loss     : boolean;        {carrier loss during xmt}
                          tx_use   : [pos(14)] _last_noerr.._using; {use status}
                          tdr      : 0..%x3fff);     {time domain reflectometry}
                 2 : { status for receive descriptors }
                        ( ovf      : boolean;        {overflow}
                          crcerr   : boolean;        {crc error}
                          frame    : boolean;        {framing alignment error}
                          short    : boolean;        {short packet}

                          rbl_hi   : [pos(8)] 0..7;  {bits 8-10 of rx byte length}
                          runt     : boolean;        {packed could not be flushed}
                          discard  : boolean;        {packet discarded}
                          esetup   : boolean;        {eloop/setup packet}
                          rx_use   : [pos(14)] _last_noerr.._using;  {use status}
                          rbl_low  : byte)           {high bits of byte length}
             end; { record }

         { Mode bits }

         mode_flag = packed record
             case boolean of
                 true  : ( full : integer );
                 false : ( multicast     : boolean;
                           promiscuous   : boolean;   {promiscuous mode}
                           led_value     : 0..3;      {LED value}
                           sanity_timer  : 0..7)      {sanity timer in 1/4 sec.}
             end; { record }

         qna_boot_structure = record
             transmit_list        : transmit_list_array;
             xmt_list_address     : phy_address;
             receive_list         : receive_list_array;
             rec_list_address     : phy_address;

             hardware_address     : datalink_address; {current system address}
             load_server_address  : datalink_address; {address of load server}

             transmit_buffer      : datalink_message;
             xmt_buffer_address   : phy_address;
             receive_buffer       : array [0..rec_list_length-1] of datalink_message;
             rec_buffer_address   : array [0..rec_list_length-1] of phy_address;

             address_control_packet : setup_descriptor;      {setup packet}
             setup_packet_physical  : phy_address;   {address of setup packet}
             end;

     qna_structure_pointer = ^qna_boot_structure;
```

```
        var

            qna_ptr          : qna_structure_pointer;
            qna_reg          : register_ptr;              {qna registers pointer}
            receive_index    : integer;

```

```
        function net_init ( var devtype        : integer;
                                xptr           : ^anytype ) : boolean;
        {++
        {
        { This procedure is called to initialize the QNA controller
        {
        { Inputs:
        {
        {       xptr = pointer to some usable memory
        {
        { Outputs:
        {
        {       devtype = qna device type
        {
        {       controller initialized
        {
        {--}
        type
            station_addr = [word] packed record
                case boolean of
                    true  : ( full : word_reg );
                    false : ( address_byte : [byte] byte );
                end; { record }

        var
            pa_byte : station_addr;
            i       : integer;
            xmt_index : integer;
            rec_index : integer;
            six_bytes : integer;
            status    : boolean;

        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


            { Save the location of the available memory }

            qna_ptr := xptr;
                                      0012
                08 A7 08 AC D0        0012            movl    08(ap),08(r7)

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


            { Initialize the register and vector addresses }

```

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

        { Initialize the controller: toggle the reset bit and boot
          diagnostic bit to reset the controller and turn on the LEDs }

        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)

        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)

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

        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)

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

        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)

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

        { Get the descriptor lists, buffers, and setup packet addresses }

        with qna_ptr^ do
                                        0054
                56 08 A7 D0             0054            movl    08(r7),r6

            begin
                                        0058

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

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

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

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

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

 2                      begin
                                       0071

 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)

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

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

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

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

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

 3                          begin
```

```
                                    009E
     4                    use_status    := _not_using;
                                    009E
              62 02 0E 02 F0        009E         insv    #2,#E,#2,(r2)
     4                    chain_addr    := false;
                                    00A3
     4                    valid         := false;
                                    00A3
              03 A2 C0 8F 8A        00A3         bicb2   #C0,03(r2)
     4                end;
                                    00A8
                 EA 53 01 F3        00A8         aobleq  #1,r3,vcg.2
     2
     2
     2      { And the receive descriptor list }
     2
     2      for rec_index := 0 to rec_list_length-1 do
                                    00AC
                       54 D4        00AC         clrl    r4
                                    00AE  vcg.3:
     2          begin
                                    00AE
     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]
     3                                  address ( receive_buffer[rec_index] );
     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
     3              begin
                                    00C8
     4              use_status  := _not_using;
                                    00C8
              63 02 0E 02 F0        00C8         insv    #2,#E,#2,(r3)
     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)
     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)
```

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

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

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

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

     4              end;
                                   00FD

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

     2
     2      { Last entry is to chain }
     2
     2      with receive_list[rec_list_length] do
                                   0101

     2          begin
                                   0101

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

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

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

     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)

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

     3          end;
                                   0122

     2
```

```
 2              { Pass the descriptor address to the controller }
 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)

 2              write_register ( qna_reg^.rx_desc_list_high, rec_list_address.high );
                                         012B
    53 5A A6 06 00 EF    012B          extzv   #0,#6,5A(r6),r3
          52 04 A7 D0    0131          movl    04(r7),r2
          06 A2 53 B0    0135          movw    r3,06(r2)

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

 2
 2              end; { with qna_ptr^ }
                                         0140

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

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

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

 1              begin
                                         0150

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

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

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

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

 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

 2               end;
                                   0178

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

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

 1               begin
                                   0182

 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

 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

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

 2                   begin
```

```
                                       0199

 3                    status := set_mode ( led3_out );
                                       0199
                           0C DD       0199          pushl   #C
                   0486 CF 01 FB       019B          calls   #1,SET_MODE

 3                    reset_for_normal_operations;
                                       01A0
                     1D AF 00 FB       01A0          calls   #0,RESET_FOR_NORMAL_OPERATIONS

 3                    devtype := xq_devtype;
                                       01A4
                     04 BC 01 D0       01A4          movl    #1,@04(ap)

 3                    net_init := true;
                                       01A8
                        58 01 90       01A8          movb    #1,r8

 3                    end
                                       01AB
                                       01AB  vcg.5:

 2
 2                end;
 1
 1        end;
                                       01AB
                        50 58 90       01AB          movb    r8,r0
                              04       01AE          ret
                                       01AF

 0
```

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

1
1              { Simply set the reset bit }
1
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)

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

0
```

```
 0
 0
 0       procedure reset_for_normal_operations;
!1       {++
!1       {
!1       { This procedure resets the device either following the initialization tests
!1       { or after a receive or transmit timeout.
!1       {
!1       {--}
 1
 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

 1
 1          { Reset the device }
 1
 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)

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

 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)

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

 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)

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

 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)

 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
```

XQB
ELN

```
0
0
0        procedure load_transmit_descriptor ( msize : integer;
1                                             buffer_address : phy_address;
1                                             setup_packet   : boolean );
1        {++
1        {
1        { This function will load the transmit descriptor
1        {
1        { Inputs:
1        {
1        {          msize specifies the size of the message
1        {          buffer_address is the physical address of the buffer
1        {          setup_packet is true if the buffer is a setup packet
1        {
1        { Outputs:
1        {
1        {          Transmit descriptor initialized
1        {
1        {--}
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)

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

1                begin
                                    0229

2                use_status  := _not_using;
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)

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)

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)

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)

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

  2          end_of_mess := true;
                               025C

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

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

  2          end;
                               0267

  1
  1      end;
                               0267
                         04    0267        ret
                               0268

  0
  0
```

```
function start_transmission : boolean;
{++
{
{ This function will write the transmit descriptor address to the qna
{ and wait for the qna to transmit the packet and receive the looped
{ response (called by the loopback routine and the set_mode routine).
{
{ Inputs:
{
{       none
{
{ Outputs:
{
{       function returns the value of the csr on return
{
{--}
var
     time_count : integer;

begin
                              0268
                              0268  START_TRANSMISSION:
                      C01C    0268          .entry  START_TRANSMISSION,^m<dv,iv,r2,r3,r4>
      54 00000000 EF 9E       026A          movab   $DATA,r4

     { Write the descriptor address }
     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)

     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)

     { Wait for completion }

     time_count := 0;
                              0290
                  53 D4       0290          clrl    r3

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

          time_count := time_count+1;
```

```
                              029A
                       53 D6  029A        incl    r3

 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:

 2             (time_count > max_wait_count);
 1
 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:

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

 0
```

```
0
0
0        function internal_locp : boolean;
1        {++
1        {
1        { This function handles the special case of internal loopback
1        {
1        {--}
1
1        var
1            time_count : integer;
1            csr        : csr_register;
1
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 9E      02C1          movab   $DATA,r2

1
1            { Write the descriptor address }
1
1            write_register (qna_reg^.tx_desc_list_low,
                                    02C8
                   50 08 A2 D0      02C8          movl    08(r2),r0
                   5C 04 A2 D0      02CC          movl    04(r2),ap
                08 AC 18 A0 B0      02D0          movw    18(r0),08(ap)

1                           qna_ptr^.xmt_list_address.low);
1            write_register (qna_reg^.tx_desc_list_high,
                                    02D5
                   5C 08 A2 D0      02D5          movl    08(r2),ap
             53 1A AC 06 00 EF      02D9          extzv   #0,#6,1A(ap),r3
                   5C 04 A2 D0      02DF          movl    04(r2),ap
                   0A AC 53 B0      02E3          movw    r3,0A(ap)

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

1            repeat
                                    02E9
                                    02E9  vcg.9:

2                csr:= read_register ( qna_reg^.csr );
                                    02E9
                   5C 04 A2 D0      02E9          movl    04(r2),ap
                   50 0E AC B0      02ED          movw    0E(ap),r0
                   FE AD 50 B0      02F1          movw    r0,-02(fp)

2                time_count := time_count+1;
```

```
                                02F5
                   53 D6        02F5           incl    r3

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

 2       until (csr.txi) or (time_count > max_wait_count);
                                02FC
             09 FE AD 07 E0     02FC           bbs     #7,-02(fp),vcg.10
       000249F0 8F 53 D1        0301           cmpl    r3,#249F0
                      DF 15     0308           bleq    vcg.9
                                030A  vcg.10:

 1
 1       { Clear the transmit bit and enable the receiver }
 1
 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)

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

 1       repeat
                                0316
                                0316  vcg.11:

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

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

 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:

 2            (time_count > max_wait_count);
 1
 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
```

```
 0
 0
 0        function loopback_test ( internal : boolean ) : boolean;
 1        {++
 1        {
 1        { This function will perform the indicated kind of loopback test (either
 1        { internal or external) and return the results of the test.
 1        {
 1        { Inputs:
 1        {
 1        {       internal is true for internal loopback test
 1        {
 1        { Outputs:
 1        {
 1        {       function returns true if test passed
 1        {
 1        {--}
 1        var
 1
 1           status     : boolean;
 1           time_count : integer;
 1           msize      : integer;
 1
 1        begin
                                    0345
                                    0345  LOOPBACK_TEST:
                              C0FC  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

 1
 1           { Set the proper value in the csr }
 1
 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)

 1                                         eloop  := not internal,
 1                                         rx_ena := false );
 1
 1
 1           { Write the descriptor address to start the transmission }
 1
 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 6E 00 2C      037F         movc5   #0,(sp),#0,#5F8,0664(r2)[r3]

 1
```

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

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

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

 1
 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

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

 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:

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

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

 1              begin
                                        03C0

 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
```

```
0
0
0        function net_receive ( var buff   : string(buff_size);
1                                   buff_size : word ) : boolean;
1
1        {++
1        {
1        { This function is called to receive a message
1        {
1        { Inputs:
1        {
1        {       buff = buffer to receive message
1        {       buff_size = buffer size
1        {
1        { Outputs:
1        {
1        {       Message received into callers' buffer
1        {
1        {--}
1
1        var
1            source_address : datalink_address;
1            dest_address   : datalink_address;
1            protocol_type  : datalink_protocol_type;
1            xlen           : integer;
1            done           : boolean;
1            timecount      : integer;
1            status         : boolean;
1
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

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

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

1
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:

1                begin
```

```
                                 042E
  2          repeat
                                 042E
  3              timecount := timecount + 1;
                                 042E
                        59 D6    042E          incl    r9
  3              nop;
                                 0430
              0277 CF 00 FB      0430          calls   #0,NOP
  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:
  3                  (timecount > max_wait_count);
  2
  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
  2
  2          { Clear the interrupt bits }
  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)
  2                                          nxm_int := true,
  2                                          rx_ena  := true,
  2                                          iloop   := true );
  2
  2          with qna_ptr^ do
                                 0466
                 58 08 A7 D0     0466          movl    08(r7),r8
  2              begin
                                 046A
  3
  3              net_receive := false;
                                 046A
                       6E 94     046A          clrb    (sp)
  3
```

```
    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

    3                    begin
                                      0482

    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)

    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

    4
    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:

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

    5
    5                       { Extract length and message out of the data buffer }
    5
    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

    5                          begin
                                      04C5

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

    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)

    6                          end; { with }
```

```
                                04D2

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

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

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

  5                        end;
                                04DE

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

  3                  else
  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

  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:

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

  3
  3                  end; { with }
                                04FB

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

  2                  begin
                                04FE

  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)

       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)

       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)

       3                     end;
                                      052B
                                      052B  vcg.29:

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

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

       0
```

```
      0
      0
      0          function net_transmit ( var buff   : string(buff_size);
      1                                  buff_size : word ) : boolean;
      1
     !1          {++
     !1          {
     !1          { This function is called to transmit a message
     !1          {
     !1          { Inputs:
     !1          {
     !1          {        buff = buffer
     !1          {        buff_size = buffer size
     !1          {
     !1          { Outputs:
     !1          {
     !1          {        buffer transmitted
     !1          {--}
      1          var
      1              csr         : csr_register;
      1              msize       : integer;
      1              timecount   : integer;
      1
      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

      1
      1              with qna_ptr^ do
                                        0541
                      56 08 A7 D0       0541          movl    08(r7),r6

      1                  begin
                                        0545

      2
      2                  with transmit_buffer do
                                        0545

      2                      begin
                                        0545

      3                      dest_address    := load_server_address;
                                        0545
                68 A6 62 A6 06 28       0545          movc3   #6,62(r6),68(r6)

      3                      source_address := hardware_address;
                                        054B
                6E A6 5C A6 06 28       054B          movc3   #6,5C(r6),6E(r6)

      3                      protocol_type  := load_protocol_type;
                                        0551
                  74 A6 0160 8F B0      0551          movw    #160,74(r6)

      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)
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)
3             end; { with }
2
2         { Load the transmit descriptor }
2
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
2         if msize < datalink_min_packet_size then
                              056F
            3C 52 D1          056F              cmpl    r2,#3C
               03 18          0572              bgeq    vcg.31
2             msize := datalink_min_packet_size;
                              0574
            52 3C D0          0574              movl    #3C,r2
                              0577  vcg.31:
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
2
2         { Write the descriptor address to start the transmission }
2
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)
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)
2
2         end; { with }
                              059B
1
1     { Wait for completion }
1
1     timecount := 0;
```

```
                              059B
                  53 D4       059B           clrl     r3
 1            repeat
                              059D
                              059D  vcg.32:
 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)
 2               timecount := timecount + 1;
                              05A9
                  53 D6       05A9           incl     r3
 2               nop;
                              05AB
         00FC CF 00 FB        05AB           calls    #0,NOP
 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:
 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)
 1                                           nxm_int := true,
 1                                           rx_ena  := true,
 1                                           iloop   := true );
 1
 1            { Set the return status.  If the transmit timed out, reset the device }
 1
 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:
 1              ( qna_ptr^.transmit_list[0].status.tx_use = _last_werr ) then
 1               net_transmit := false
                              05E1
                  52 94       05E1           clrb     r2
                  03 11       05E3           brb      vcg.36
                              05E5  vcg.35:
 2            else
```

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

     1
     1          { Load all addresses as the hardware address }
     1
     1          with qna_ptr^ do
                                     05F5
                        52 08 A7 D0  05F5            movl    08(r7),r2
     1              for i := 1 to 2 do
                                     05F9
                           58 01 D0  05F9            movl    #1,r8
                                     05FC  vcg.37:
     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:
     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:
     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
     5                                     hardware_address::datalink_address_byte[j];
     1
     1      end;
                                     0625
                                 04  0625            ret
                                     0626

     0
```

```
 0
 0
 0        function set_mode ( led_value_mode : integer ) : boolean;
 1        {++
 1        {
 1        { This procedure will write the mode to the qna
 1        {
 1        { Inputs  : Current mode flags are used to set the mode
 1        {
 1        { Outputs : Controller mode set
 1        {
 1        {--}
 1        var
 1            mode_multiplier : integer;
 1            tsize           : integer;
 1            time_count      : integer;
 1
 1        begin
                              0626
                              0626  SET_MODE:
                        C00C  0626          .entry   SET_MODE,^m<dv,iv,r2,r3>
         52 00000000 EF 9E    0628          movab    $DATA,r2

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

 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

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

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

 1                begin
                              0654

 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
```

```
0
0
0        procedure delay;
1
1        var
1
1            i : integer;
1
1        begin
                                         069A
                                         069A   DELAY:
                                  C000   069A            .entry  DELAY,^m<dv,iv>

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

1                nop
                                         069F
                        09 AF 00 FB      069F            calls   #0,NOP
              F4 5C 000001F4 8F F3       06A3            aobleq  #1F4,ap,vcg.42
                                    04   06AB            ret
                                         06AC

2
2        end;
0
0
0        procedure nop;
1        begin
                                         06AC
                                         06AC   NOP:
                                  C000   06AC            .entry  NOP,^m<dv,iv>

1        end;
                                         06AE
                                    04   06AE            ret
                                         06AF

0
0
0        end; { module xqboot }
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