DP839EB-ATS SONIC Packet Driver for PC/TCP by FTP Software

National Semiconductor
Application Note 748
January 1991

INTRODUCTION

This is a program listing for a driver for the DP839EB-ATS SONIC Ethernet Adapter. This driver enables the DP839EB-ATS to operate with a TCP/IP software package from FTP Software Inc. called PC/TCP. This driver is written to version 2.0x of this software package.

This software program listing is provided primarily as a programming example for writing software for the DP83932 Systems Oriented Network Interface Controller. This driver is written in Microsoft C 5.1 and Microsoft Assembler 5.1. Since the bulk of the software is written in C, the concepts provided are easily portable to other environments.

This example driver was not written to achieve optimum performance with PC/TCP, but primarily to show how the SONIC Controller can be programmed.

This software does not make use of higher performance upper level features, and performance is limited by this.

The driver is listed by modules in the order listed below.

1. pktdrv.c
2. far.c
3. isr.c
4. sonic.c
5. pktdrv.h
6. sonic.h
7. isrlib.asm
8. pktint.asm
9. pktdrv.mak (make file)

FILENAME: pktdrv.c

```
static char Pktdrv_Sid[]      = "%W%      %G%";
/*
******************************************************************************
*         Copyright (c) 1990 by National Semiconductor Corporation          *
*                           All Rights Reserved                             *
******************************************************************************

==============================================================================
FILE:        pktdrv.c
NOTES:       This program is a packet driver that provides a common interface
             between PC/TCP's kernel and NSC's SONIC hardware. This program
             was based on a set of drivers provided by Clarkson from FTP.
             This driver is NOT for performance testing due to PC/TCP limitations.
==============================================================================
UPDATE LOG:
When/Who                Why/What/Where
------------------------------------------------------------------------------
10/23/90 Mike Lui       Convert to work for SONIC

==============================================================================
*/

#include <stdio.h>
#include <dos.h>
#include <memory.h>
#include <string.h>
#include "pktdrv.h"
#include "sonic.h"

  /* externals */
extern void (interrupt far drv_isr)();    /* the interrupt we use */
extern unsigned _psp;               /* segment address of PSP */
```

TL/F/11142–1

Microsoft® is a registered trademark of Microsoft Corporation.
PC/TCP® is a registered trademark of FTP Software Inc.

```
  /* Driver information */
static unsigned int     drv_version = 1;  /* driver version */
static unsigned char    drv_class = 1;          /* driver class */
static unsigned int     drv_type = 14;          /* driver type */
static unsigned char    drv_number = 0;         /* driver number */
static unsigned int     drv_funct = 1;          /* basic driver function */
static char drv_name[] =            /* driver name */
    "National Semiconductor SONIC/TCP Packet Driver";
static char cpy_msg[] =
    "Copyright (c) 1990 by National Semiconductor Corporation";
static char drv_rev[] = "1.0";            /* current driver rev */

static HANDLE handle_tbl[MAX_HANDLES];    /* create handle structs */
void (interrupt far *sys_isr)();    /* remember system isr */
char far *pkt_signature = "PKT DRVR";
unsigned int packet_int_no = 0x60;  /* interrupt for communications */
static unsigned far *psp_ptr;       /* pointer to PSP */
unsigned mem_sz;              /* program memory size in paragraphs */

union REGS r_regs;
struct SREGS s_regs;
int send_pending;             /* required for Synernetics */
static int syn_installed;           /* required for Synernetics */

extern int  opterr;
extern int  optind;
extern char *optarg;

/*
 * main()
 *
 *  Main procedure.
 *  Once initialization is complete terminate and stay resident.
 */
main(argc, argv)
int argc;
char *argv[];
{
    psp_ptr = (unsigned far *)((unsigned long)_psp << 16);
    mem_sz = (psp_ptr[1] - _psp);

    init_drv(argc, argv);           /* initialize driver and hardware */

    outp(pagebase, 0);
    outpw(regbase+cr, 8);         /* enable receiver */

    /* terminate and stay resident */
    _dos_keep(0, mem_sz);

}


/*
 * int_handler()
 *
 *  This routine is called from an assembly isr routine "drv_isr"
 *  to handle the application interrupt. The isr routine passes a
 *  set of pointers of the registers to this routine. Register AH
 *  contains which function is to be performed. These registers will
 *  be restored in "drv_isr" before returning from the interrupt.
 *
```
TL/F/11142–2

```
 *    Return values:    If an error occurred the value will be in
 *                the DH register and the carry bit of cflag
 *                will be set.
 */
int_handler(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    int ret_val;

    switch(regs->h.ah) {
    case 1:
      ret_val = driver_info(regs, sregs);
      break;
    case 2:
      ret_val = access_type(regs, sregs);
      break;
    case 3:
      ret_val = release_type(regs, sregs);
      break;
    case 4:
      ret_val = send_packet(regs, sregs);
      break;
    case 5:
      ret_val = terminate(regs, sregs);
      break;
    case 6:
      ret_val = get_address(regs, sregs);
      break;
    case 7:
      ret_val = reset_interface(regs, sregs);
      break;
    case 24:
      ret_val = get_stats(regs, sregs);
      break;
    default:
      ret_val = BAD_COMMAND;
    }

    if(ret_val) {
      regs->h.dh = ret_val;               /* put error code into dh */
      regs->x.cflag |= 0x1;               /* and set carry bit */
    }
}

/*
 * driver_info()
 *
 *  Return information on the driver interface. Handle is optional
 *  and is not used in new driver??
 *
 *    Return values:  0 - Success
 */
driver_info(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    regs->x.bx = drv_version;             /* driver version */
    regs->h.ch = drv_class;               /* driver class */
    regs->x.dx = drv_type;                /* driver type */
    regs->h.cl = drv_number;              /* driver number */
    regs->x.si = (unsigned)drv_name;            /* driver name */
```

TL/F/11142–3

```
    sregs->ds  = (unsigned long)((char far *)drv_name) >> 16;
    regs->h.al = drv_funct;              /* driver function */
    return 0;
}


/*
 * access_type()
 *
 *  Initiate access to packets for the specific type. Since the packet
 *  type field needs to have the bytes of 16 bit values swaped, the
 *  handle will store the type field byte swapped.
 *
 *    Return values:     0 - Success
 *                >0 - Failure
 */
access_type(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    int  i, n,
       open_handle = OPEN,               /* available handle */
       type_cnt;
    unsigned char type_buf[MAX_TYPE_LEN];

    /* first check a few things to make sure packet access is ok */

    /* check class */
    if(regs->h.al != drv_class) {
      return NO_CLAS;
    }

    /* check type (ours or generic) */
    if(!((regs->x.bx == drv_type) || (regs->x.bx == -1))) {
      return NO_TYPE;
    }

    /* check number (ours or generic) */
    if(!((regs->h.dl == 0) || (regs->h.dl == 1))) {
      return NO_NUMBER;
    }

    /* check packet type length, if too long its not ours */
    if(regs->x.cx > MAX_TYPE_LEN) {
      return TYPE_INUSE;
    }

    /*
     * now check for an available handle and if the handle already
     * exists with same packet type.
     */
    type_ptr = (char far *)(((unsigned long)sregs->ds << 16) | regs->x.si);

    for(i = 0; i < regs->x.cx; i++)
      type_buf[i] = type_ptr[i];

    for(n = 0; n < MAX_HANDLES; n++) {
      if(handle_tbl[n].in_use) {         /* check packet type */
          type_cnt = MIN(regs->x.cx, handle_tbl[n].len);
          if(!far_memcmp((char far *)type_buf,
                        (char far *)handle_tbl[n].type, type_cnt))
            return TYPE_INUSE;           /* duplicate types */
```

TL/F/11142–4

```
        }
        else if(open_handle == OPEN)
            open_handle = n;                /* grab first open handle */
    }

    if(open_handle == OPEN)
      return BAD_HANDLE;                    /* no available handles */

    /* copy the handle */
    handle_tbl[open_handle].in_use++;

    for(i = 0; i < regs->x.cx; i++) {
      handle_tbl[open_handle].type[i] = type_buf[i];
    }
    handle_tbl[open_handle].len    = regs->x.cx;
    handle_tbl[open_handle].rec_es = sregs->es;
    handle_tbl[open_handle].rec_di = regs->x.di;

    regs->x.ax = open_handle;               /* return handle */
    return 0;                               /* return success */

}
/*
 * release_type()
 *
 *  Release access to packets with a particular handle.
 *
 *    Return values:     0 - Success
 *                >0 - Failure
 */
release_type(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    if(chk_handle(regs->x.bx))
      return BAD_HANDLE;

    /* release handle */
    handle_tbl[regs->x.bx].in_use = 0;
    return 0;
}

/*
 * send_packet()
 *
 *  Send packet buffer.
 *
 *    Return values:     0 - Success
 *                >0 - Failure
 */
send_packet(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    char far *frame_ptr;                    /* pointer to frame */
    unsigned long pkt_addr;                       /* physical address of packet */
    unsigned int buf_len;                   /* frame length */
    int i;

    short previous_tda;

    /* check if frame is too big */
```

TL/F/11142–5

```
if(regs->x.cx > BUF_SZ) {
  return NO_SPACE;
}

/* update driver stats */
drv_stats.packets_out++;
drv_stats.bytes_out += regs->x.cx;

/* point to the app's send frame */
frame_ptr = (char far *)(((unsigned long)sregs->ds << 16) |
                               regs->x.si);
pkt_addr = (unsigned long) sregs->ds * 16 + regs->x.si;

buf_len = regs->x.cx;                 /* frame+FC+SNAP length */


/* save current tda */
previous_tda=curtda;

if (transmitactive) {
    /* network is currently busy transmitting, just queue up the tda */
    if (curtda==TDANUM-1) {
    /* load tda with the transmit fragment */
        tda[0].pkt_size=buf_len;
        tda[0].frag_count=1;
        tda[0].frag_ptr0=(unsigned short) pkt_addr;
        tda[0].frag_ptr1=pkt_addr >> 16;
        tda[0].frag_size=buf_len;
        tda[0].link |= 1;
        curtda=0;
    }
    else {
    /* load tda with the transmit fragment */
        tda[curtda+1].pkt_size=buf_len;
        tda[curtda+1].frag_count=1;
        tda[curtda+1].frag_ptr0=(unsigned short) pkt_addr;
        tda[curtda+1].frag_ptr1=pkt_addr >> 16;
        tda[curtda+1].frag_size=buf_len;
        tda[curtda+1].link |= 1;
        curtda++;
    }
    tda[previous_tda].link &= 0x0fffe;
}
else {
    /* network is free */
    retry=0;
    /* load tda with the transmit fragment */
    tda[0].pkt_size=buf_len;
    tda[0].frag_count=1;
    tda[0].frag_ptr0=(unsigned short) pkt_addr;
    tda[0].frag_ptr1=pkt_addr >> 16;
    tda[0].frag_size=buf_len;
    tda[0].link |= 1;
    curtda=0;
    outp(pagebase, 0);
    outpw(regbase+ctda, tda_addr);  /* load ctda */
    transmitactive=1;               /* set network flag to busy */
}
outp(pagebase, 0);         /* get the first page */
outpw(regbase+cr, 2);           /* issue the transmit command */

return 0;
```

TL/F/11142–6

```
}

/*
 * terminate()
 *
 *  Terminate the driver.
 *
 *    Return values:     0 - Success
 *                 >0 - Failure
 */
terminate(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    int sonic_irq;

    sonic_irq=3;

    _dos_setvect(packet_int_no, sys_isr); /* put back system isr */

    sonic_isr_disable(sonic_irq);                       /* remove sonic interrupt
*/
    /* free environment memory */
    _dos_freemem(psp_ptr[0x16]);

    /* free memory and return to app */
    if(_dos_freemem(_psp))
      return CANT_TERMINATE;

    return 0;
}


/*
 * get_address()
 *
 *  Get the local net address.
 *
 *    Return values:     0 - Success
 *                 >0 - Failure
 */
get_address(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    char buf[6];
    int i, old_mode;
    char far *addr_ptr;                    /* pointer to address */

    if(chk_handle(regs->x.bx))
      return BAD_HANDLE;

    /* get buffer */
    addr_ptr = (char far *)(((unsigned long)sregs->es << 16) | regs->x.di);

    /*
     * copy ethernet address from hardware.
     * regs->x.cx is the length of buffer, fail if address
     * is too big to fit in buffer - NO_SPACE
     */
    if(regs->x.cx < 6)
      return NO_SPACE;
```

TL/F/11142–7

```
    regs->x.cx = 6;

    for(i = 0; i < regs->x.cx; i++) {
      addr_ptr[i] = inp(iobase+i);
    }

    return 0;
}


/*
 * reset_interface()
 *
 *  Reset the interface for the particular handle. If more than one
 *  handle is open return CANT_RESET so other applications (handles)
 *  will not get confused.
 *
 *    Return values:     0 - Success
 *                >0 - Failure
 */
reset_interface(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    char far *addr_ptr;                   /* pointer to address */
    int          i, handle_cnt = 0;

    if(chk_handle(regs->x.bx))
      return BAD_HANDLE;

    /* check if there is more than one handle is open */
    for(i = MIN_HANDLE; i <  MAX_HANDLES; i++)
        if(handle_tbl[i].in_use != 0) handle_cnt++;
    if(handle_cnt > 1)
      return CANT_RESET;

    /* Reset the hardware to a known state */
    /* Will need something maybe ??? */

    return 0;
}


/*
 * get_stats()
 *
 *  Return driver statistics.
 *
 *    Return values:     0 - Success
 *                >0 - Failure
 */
get_stats(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    if(chk_handle(regs->x.bx))
      return BAD_HANDLE;

    regs->x.si = (unsigned)&drv_stats;          /* driver stats */
    sregs->ds  = (unsigned long)((char far *)&drv_stats) >> 16;
```

TL/F/11142–8

```
    return 0;
}


/*
 * drv_rcvr()
 *
 *  Receiver procedure. Once a frame is recieved, we need to make two upcall
 *  with the receiving routine provided by the application. The first
 *  call (AX == 0) is to request a buffer to copy the frame to. The second
 *  call (AX == 1) indicates that the frame has been copied.
 *
 *    Return values:     0 - Success
 *                >0 - Failure
 */
/* void far drv_rcvr()  */
drv_rcvr()
{
    int i;
    int handle_found = OPEN;        /* set if valid frame recieved */
    unsigned char far *frame;
    char far *cp_ptr;


    /* get the frame */
    while (rda[currda].status != 0) {
        frame=(unsigned char far *) (((unsigned long) rda[currda].pkt_ptr1 << 28)
| rda[currda].pkt_ptr0);
      /* validate the received frame */
      for(i = MIN_HANDLE; i < MAX_HANDLES; i++) {
          if((handle_tbl[i].in_use == 0) ||
                        ((((unsigned long)handle_tbl[i].rec_es << 16) |
                                    handle_tbl[i].rec_di) == NULL))
            continue;         /* go to next handle */
          if(!far_memcmp((char far *)handle_tbl[i].type,
                  &frame[ETYPE_OFS], handle_tbl[i].len)) {
            handle_found = i;
            break;
          }
      }
      if(handle_found == OPEN) {
          drv_stats.packets_dropped++;
            free_rda();
          continue;
      }

        /* update driver stats */
      drv_stats.packets_in++;
      drv_stats.bytes_in += rda[currda].byte_count;

      /* first upcall, tell them frame size */
      app_recv(0, handle_found, MAX(rda[currda].byte_count-4, 64),
                  (char far *)&cp_ptr, handle_tbl[handle_found].rec_di,
                  handle_tbl[handle_found].rec_es);

      /* check if copy is permitted */
      if(cp_ptr == NULL) {
          drv_stats.packets_dropped++;
            free_rda();
          continue;
      }
```

TL/F/11142–9

```
        /* copy the frame */
        far_memcpy(&cp_ptr[0], &frame[0], rda[currda].byte_count-4);

        /* second upcall, tell them frame has been copied */
        app_recv(1, handle_found, rda[currda].byte_count-4, (char far *)&cp_ptr,
               handle_tbl[handle_found].rec_di,
               handle_tbl[handle_found].rec_es);

        /* free rda */
          free_rda();
    }
    return;
}



/*
 * free_rda()
 *
 * This routine is to free up the currently examined rda for later use
 *
 */

free_rda()
{
    static int first;
    unsigned short tmp_value;

    /* check fifo overrun */
    outp(pagebase, 0);
    if (inpw(regbase+isr) & ISR_RFO)
        outpw(regbase+isr, ISR_RFO);

    /* reinitialize the rda */
    rda[currda].status=0;
    rda[currda].byte_count=0;
    rda[currda].pkt_ptr0=0;
    rda[currda].pkt_ptr1=0;
    rda[currda].in_use=0x0ffff;
    rda[currda].pkt_link |= 1;

    /* link the previous rda to the current rda */
    if (currda==0)
        rda[RDANUM-1].pkt_link&=0x0fffe;
    else
        rda[currda-1].pkt_link&=0x0fffe;

    /* get the first buffer number */
    if (!first) {
        previous_seqno=rda[currda].seq_no >> 8;
        first=1;
    }

    /* check whether rba can be reused */
    if (rda[currda].seq_no >> 8 != previous_seqno) {
        previous_seqno=rda[currda].seq_no >> 8;
        tmp_value=rwp_table[cur_rwp];
        if (cur_rwp==2)
            cur_rwp=0;
        else
            cur_rwp++;
        outp(pagebase, 0x18);
```

TL/F/11142–10

```
        outpw(regbase + rwp, tmp_value);
        outp(pagebase, 0);
        tmp_value=inpw(regbase + isr);
        if (tmp_value & ISR_RBE)
            outpw(regbase + isr, ISR_RBE);
    }

    /* check rde */
    outp(pagebase, 0);
    if (inpw(regbase+isr) & ISR_RDE) {
        outpw(regbase+isr, ISR_RDE);
        outp(pagebase, 0x0d);
        tmp_value=inpw(regbase+crda) & 0x0fffe;
        outpw(regbase+crda, tmp_value);
    }

    if (currda == RDANUM-1)
        currda=0;
    else
        currda++;

}


/*
 * init_drv()
 *
 *  Initialize the driver and hardware.
 */
init_drv(argc, argv)
int argc;
char *argv[];
{
    char far *ptr;
    int kill_drv;

    fprintf(stderr,
      "%s -- Version %s\n%s\n", drv_name, drv_rev, cpy_msg);

    kill_drv = do_args(argc, argv);       /* process command line */

    sys_isr = _dos_getvect(packet_int_no);      /* get system isr */
    ptr = (char far *)sys_isr + 3;

    if(kill_drv)                       /* terminate active driver */
      kill_driver(ptr);

    if((ptr != NULL) && (far_strcmp(ptr, pkt_signature) == 0)) {
      fprintf(stderr,
            "Error: a packet driver already exist at interrupt 0x%x\n",
            packet_int_no);
      exit(1);
    }


    _dos_setvect(packet_int_no, drv_isr); /* install driver isr */

    init();         /* init SONIC */

    fprintf(stderr,
      "Packet Driver is using INT 0x%x and %ld bytes of memory\n",
       packet_int_no, (unsigned long)mem_sz * 16);
```

TL/F/11142–11

```
}

/*
 * chk_handle()
 *
 *  Check if handle is valid.
 *
 *    Return values:     0 - Success
 *                >0 - Failure
 */
chk_handle(handle)
unsigned int handle;
{
    /* check if handle is in range */
    if((handle < MIN_HANDLE) || (handle >= MAX_HANDLES))
      return BAD_HANDLE;

    /* check if handle is in use */
    if(handle_tbl[handle].in_use == 0)
      return BAD_HANDLE;

    return 0;
}

/*
 * kill_driver()
 *
 *  Terminate driver from memory
 *
 *    Return values:     none - exits from program
 */
kill_driver(ptr)
char far *ptr;
{
    if((ptr == NULL) || (far_strcmp(ptr, pkt_signature) != 0)) {
      fprintf(stderr,
            "Error: no packet driver at interrupt 0x%x\n",
            packet_int_no);
      exit(1);
    }
    r_regs.h.ah = 5;
    r_regs.x.bx = 0;
    int86(packet_int_no, &r_regs, &r_regs);
    if(r_regs.x.cflag) {
      fprintf(stderr, "Error: packet driver can not terminate\n");
      exit(1);
    }
    printf("Terminated packet driver at interrupt 0x%x\n", packet_int_no);
    exit(0);
}



/*
 * do_args()
 *
 *  Process program arguments using getopt().
 *
 *    Return values:     0 - Success
 *                 1 - Terminate driver
 */
do_args(argc, argv)
```

TL/F/11142–12

```
int  argc;
char *argv[];
{
    int in, done = 0;
    char *sptr;

    if(argc == 1)             /* use default packet_int_no */
      return 0;

#ifdef MSDOS
    if((sptr = strrchr(*argv, '\\')) != NULL)
      strcpy(*argv, sptr + 1);
    if((sptr = strrchr(*argv, '.')) != NULL)
      *sptr = '\0';
#endif

    while (!done && ((in = getopt(argc, argv, "?hi:t:")) != -1)) {
      switch(in) {
      case 'i':
      case 't':
          if(sscanf(optarg, "0x%x", &packet_int_no) != 1)
            if(sscanf(optarg, "%d", &packet_int_no) != 1) {
                done = 1;
                break;
            }
          /*
          if(!strncmp(optarg, "0x", 2))
            sscanf(&optarg[2], "%x", &packet_int_no);
          else
            sscanf(optarg, "%d", &packet_int_no);
          */
          if((packet_int_no < 0x60) || (packet_int_no > 0x80)) {
            fprintf(stderr,
            "Error: packet_int_no should be in the range 0x60 to 0x80\n");
            exit(1);
          }
          done = 1;
          if(optind == argc) {
            if(in == 't')
                return 1;
            else
                return 0;
          }
          break;
      }
    }
    fprintf(stderr,
      "Usage: %s [-h] [-i packet_int_no] [-t packet_int_no]\n", *argv);
    fprintf(stderr,
      "  -h = this help message\n");
    fprintf(stderr,
      "  -i = set packet interrupt number, default is 0x60\n");
    fprintf(stderr,
      "  -t = terminate packet driver\n");
    exit(1);
}

int  opterr = 1;
int  optind = 1;
char *optarg;
/*
 * getopt() -- Gets options from command line and breaks them up for analysis.
```

TL/F/11142–13

```
 *            It is functionally compatible with the UNIX version.
 * By Ted Thi
 */
getopt(argc, argv, ctrlStr)
int  argc;
char **argv,
     *ctrlStr;
{
  extern char   *strchr();
  register char *s_ptr;
  static int    i;
  if (optind < argc && argv[optind][++i] == '\0') {
    if (i == 1 || ++optind >= argc)
      return(-1);
    i = 1;
  }
  if (i <= 1) {
    if (optind >= argc || (*argv[optind] != '-' && *argv[optind] != '/') ||
      argv[optind][1] == '\0')
      return(-1);
    if (strcmp(argv[optind] + 1, "-") == 0) {
      optind++;
      return(-1);
    }
  }
  if (argv[optind][i] == ':' || (s_ptr = strchr(ctrlStr, argv[optind][i]))
      == NULL) {
    if (opterr)
      fprintf(stderr, "%s: illegal option -- %c\n", *argv, argv[optind][i]);
    return('?');
  }
  if (s_ptr[1] == ':') {
    if (argv[optind][++i] == '\0') {
      i = 0;
      if (++optind >= argc) {
        if (opterr)
          fprintf(stderr, "%s: option requires an argument -- %c\n", *argv,
                  *s_ptr);
        return('?');
      }
    }
    optarg = argv[optind++] + i;
    i = 0;
  } else
    optarg = NULL;
  return(*s_ptr);
}                                    /* of getopt() */
```

TL/F/11142–14

FILENAME: far.c

```
/*
 ******************************************************************************
 *           Copyright (c) 1990 National Semiconductor Corporation            *
 *                        All Rights Reserved                                 *
 ******************************************************************************
 */
#include <dos.h>


void far_memcpy(dest, src, cnt)
register char far *dest;
register char far *src;
register unsigned cnt;
{
  while (cnt--) *dest++ = *src++;
}


char far *far_strcpy(s1, s2)
register char far *s1, far *s2;
{
    char far *s3 = s1;
    while (*s2) *s1++ = *s2++;
    return (s3);
}


far_strcmp(s1, s2)
register char far *s1, far *s2;
{
    while(*s1) {
      if(*s1 != *s2) return(*s1 - *s2);
      s1++; s2++;
    }
    return(*s1 - *s2);
}


far_memcmp(s1, s2, cnt)
register char far *s1, far *s2;
register int cnt;
{
    while(--cnt > 0) {
      if(*s1 != *s2)
          return(*s1 - *s2);
      s1++; s2++;
    }
    return(*s1 - *s2);
}
```

TL/F/11142–15

FILENAME: isr.c

```
/*
 ****************************************************************************
 *           Copyright (c) 1990 National Semiconductor Corporation          *
 *                          All Rights Reserved                             *
 ****************************************************************************
 */

#include <dos.h>
#include "sonic.h"

#define ISR_STACK_SZ    2048

static char irq_map[] = {
    0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77
};

static int pic_ctl;
static int pic_mask;
static int old_mask_val;

void (interrupt far *sys_irq_int)();

void interrupt far sonic_isr();

void sonic_isr_enable(irq)
int irq;
{
    pic_ctl = irq < 8 ? 0x20 : 0xa0;
    pic_mask = pic_ctl + 1;

    old_mask_val = inp(pic_mask);
    sys_irq_int = _dos_getvect(irq_map[irq]);

    _disable();
    _dos_setvect(irq_map[irq], sonic_isr);
    outp(pic_mask, old_mask_val & ~(1 << irq));
    _enable();
}


void sonic_isr_disable(irq)
int irq;
{
    _disable();
    _dos_setvect(irq_map[irq], sys_irq_int);
    outp(pic_mask, old_mask_val);
    _enable();
}


static char far *old_sp;
static char isr_stack[ISR_STACK_SZ];


void interrupt far sonic_isr()
{
    char far *(far get_sp)();
    void (far set_sp)();
    unsigned short isr_reg;
```

TL/F/11142–16

```
    unsigned short activetda;

    outp(pagebase, 0);
    outpw(regbase+imr, 0);          /* unmask the imr */

    old_sp = get_sp();
    set_sp((char far *)isr_stack + ISR_STACK_SZ);

    _enable();

    outp(pagebase, 0);           /* get the right page */
    isr_reg=inpw(regbase+isr);

    while (isr_reg) {
        if (isr_reg & ISR_PKTRX)  {         /* is there a receive */
            outp(pagebase, 0);
            outpw(regbase+isr, ISR_PKTRX);  /* clear receive bit */
            drv_rcvr();                     /* process rda */
        }
        if (isr_reg & ISR_TXDN) {           /* is there is transmit done */
            outp(pagebase, 0);           /* clear transmit done bit */
            outpw(regbase+isr, ISR_TXDN);

            transmitactive=0;
        }
        if (isr_reg & ISR_TXER) {           /* is there a transmit error */
            outp(pagebase, 0);           /* clear transmit error bit */
            outpw(regbase+isr, ISR_TXER);
            if (retry > 10) {               /* if retry 10 and still not succeed
to transmit this tda */
                outp(pagebase, 0);       /* throw away this tda */
                activetda=inpw(regbase+ctda);
                activetda &= 0x0fffe;
                outpw(regbase+ctda, activetda+16);
            }
            else {                          /* try again */
                outp(pagebase, 0);
                outp(regbase+cr, 2);        /* transmit */
            }
        }
        outp(pagebase, 0);
        isr_reg=inpw(regbase+isr);
    }

    _disable();

    set_sp(old_sp);

    outp(pic_ctl, 0x20);

    outp(pagebase, 0);
    outpw(regbase+imr, 0x0700);

}
```

TL/F/11142–17

FILENAME: sonic.c

```
#include "sonic.h"
#include "dos.h"

/*
 * init()
 *
 * This routine is from init_drv() to initialize sonic buffer and sonic
 * registers.
 *
 * Return values: 0 if success
 *                1 if fail
 *
 */

init()
{
    short i;
    unsigned short  cur_loc;
    int sonic_irq;

    /* set up DMA controller */
    outp(0xd0, 0x10);
    outp(0xd6, 0xd2);
    outp(0xd4, 0x02);
    outp(0xde, 0x00);

    /* initialize valuables */
    transmitactive=0;
    curtda=0;
    currda=0;
    sonic_irq=3;
    /* install sonic interrupt */
    sonic_isr_enable(sonic_irq);

    /* initialize sonic register */
    outp(pagebase, 0);          /* set the right page */
    outpw(regbase+cr, 0x94);         /* reset sonic */
    outpw(regbase+dcr, 0x12de);     /* set configuration: 3 wait state
                                                        16-bit data path
                                                        block mode
                                                        8 words receive fifo
                                                        12 words transmit fifo */
    outpw(regbase+cr, 0);           /* out of reset mode */
    outpw(regbase+rcr, 0x2000);
    outpw(regbase+isr, 0x0ffff);  /* reset isr */
    outpw(regbase+imr, 0x700);      /* set mask to xmit done, xmit error and
                                     receive packet */

    init_tda();                   /* init tda */
    init_rda();                   /* init rda */
    init_rra();                   /* init rra */
    init_cam();                   /* init cam */

    /* initialize rwp location table */
    outp(pagebase, 0x15);
    cur_loc=inpw(regbase+rsa);
    for (i=0; i<3; i++) {
        rwp_table[i]=cur_loc;
        cur_loc+=8;
```

TL/F/11142–18

```
    }
    cur_rwp=0;

    /* normal operation */
    outp(pagebase, 0);
    outpw(regbase+cr, 0x100);      /* read rra */

    return(0);
}

/*
 * init_tda()
 *
 * This routine is to link the tda so as to make transmission more
 * efficient. It also initialize the utda and ctda registers.
 *
 */

init_tda()
{
    unsigned short i, u16, l16;
    unsigned long addr32;
    char far *ptr;
    struct SREGS segregs;

    segread(&segregs);      /* Read the segment register value */
    /* link the first nine tda */
    for (i=0; i<TDANUM-1; i++) {
      addr32=(((unsigned long) segregs.ds << 16) | ((unsigned short) &tda[i+1]));
        u16=addr32>>16;
        l16=(unsigned short)addr32;
        addr32=(unsigned long)u16 * 16 + l16;
        tda[i].config=0x1000;
        tda[i].link=(unsigned short) addr32;
    }

    /* set the last tda link field to the first tda */
    addr32=(((unsigned long) segregs.ds << 16) | ((unsigned short) &tda[0]));
    u16=addr32>>16;
    l16=(unsigned short)addr32;
    addr32=(unsigned long)u16 * 16 + l16;
    tda[TDANUM-1].link=(unsigned short) addr32;

    /* set the utda and ctda register */
    outp(pagebase,0);                /* get the correct page */
    outpw(regbase+utda, addr32>>16);      /* set utda */
    outpw(regbase+ctda, (unsigned short)addr32);          /* set ctda */
    tda_addr=(unsigned short)addr32;
}




/*
 * init_rda()
 *
 * This routine is to link the rda together. It also initialize the urda and
 * crda registers.
 *
 */
```

TL/F/11142–19

```
init_rda()
{
    unsigned short i, u16, l16;
    unsigned long addr32;
    struct SREGS segregs;

    segread(&segregs);      /* Read the segment register value */

    /* link the rda */
    for (i=0; i<RDANUM-1; i++) {
        addr32=(((unsigned long) segregs.ds << 16) | ((unsigned short)
&rda[i+1]));
        u16=addr32>>16;
        l16=(unsigned short)addr32;
        addr32=(unsigned long)u16 * 16 + l16;
        rda[i].pkt_link=(unsigned short) addr32;
        rda[i].in_use=0x0ffff;
    }

    /* set the last rda link field to the first rda */
    addr32=(((unsigned long) segregs.ds << 16) | ((unsigned short) &rda[0]));
    u16=addr32>>16;
    l16=(unsigned short)addr32;
    addr32=(unsigned long)u16 * 16 + l16;
    rda[RDANUM-1].in_use=0x0ffff;
    rda[RDANUM-1].pkt_link=(unsigned short) addr32;
    rda[RDANUM-1].pkt_link|=1;                 /* set EOL */

    /* set the urda and crda register */
    outp(pagebase,0x0d);                /* get the correct page */
    outpw(regbase+urda, addr32>>16);      /* set urda */
    outpw(regbase+crda, (unsigned short)addr32);          /* set crda */

}

/*
 * init_rra()
 *
 * This routine is initialize the rra and set rsa, rea, rrp, rwp registers
 *
 */

init_rra()
{
    unsigned short i, u16, l16;
    unsigned long addr32;
    struct SREGS segregs;

    segread(&segregs);      /* Read the segment register value */

    /* initialize the rra slot */
    for (i=0; i<RRANUM; i++) {
        addr32=(((unsigned long) segregs.ds << 16) | ((unsigned short) &rba[i]));
        u16=addr32>>16;
        l16=(unsigned short)addr32;
        addr32=(unsigned long)u16 * 16 + l16;
        rra[i].buff_ptr0=(unsigned short)addr32;
        rra[i].buff_ptr1=addr32>>16;
        rra[i].buff_wc0=0x800;
        rra[i].buff_wc1=0;
    }
```

TL/F/11142–20

```
    addr32=(((unsigned long) segregs.ds << 16) | ((unsigned short) &rra[0]));
    u16=addr32>>16;
    l16=(unsigned short)addr32;
    addr32=(unsigned long)u16 * 16 + l16;

    /* set urra, rsa, and rrp */
    outp(pagebase, 0x14);        /* set the right page */
    outpw(regbase+urra, addr32 >> 16);    /* set urra */
    outpw(regbase+rsa, (unsigned short)addr32);           /* set rsa */
    outpw(regbase+rrp, (unsigned short)addr32);           /* set rrp */

    /* set rea and rwp */
    addr32+=24;
    outpw(regbase+rea, (unsigned short) addr32);          /* set rea */
    outp(pagebase, 0x18);
    outpw(regbase+rwp, (unsigned short) addr32);          /* set rwp */
}

/*
 * init_cam()
 *
 * This routine is initialize the cam and set cdp, cdc registers. Also,
 * load the cam.
 *
 */

init_cam()
{
    unsigned short i, u16, l16;
    unsigned long addr32;
    struct SREGS segregs;

    segread(&segregs);      /* Read the segment register value */
    addr32=(((unsigned long) segregs.ds << 16) | ((unsigned short) &cam));
    u16=addr32>>16;
    l16=(unsigned short)addr32;
    addr32=(unsigned long)u16 * 16 + l16;

    outp(pagebase, 0x26);
    outpw(regbase+cdp, (unsigned short) addr32);    /* load cdp */
    outpw(regbase+cdc, 16);                          /* load cdc */

    /* load the cda with node physical address */
    cam.cam_port_info[0].port0=inpw(iobase);
    cam.cam_port_info[0].port1=inpw(iobase+2);
    cam.cam_port_info[0].port2=inpw(iobase+4);
    for(i=0; i<16; i++)
        cam.cam_port_info[i].entry_ptr=i;
    cam.cam_enable=1;                               /* load cam enable */
    /* load cam */
    outp(pagebase, 0);
    outpw(regbase+cr, CMD_LCAM);
    /* to ensure load cam is properly executed and clear LCD bit in isr */
    for (;;) {
        if (inpw(regbase+isr) & ISR_LCD) {
            outpw(regbase+isr, ISR_LCD);
            break;
        }
    }
}
```

TL/F/11142–21

FILENAME: pktdrv.h

```
/*
******************************************************************************
*         Copyright (c) 1990 by National Semiconductor Corporation          *
*                          All Rights Reserved                              *
******************************************************************************
*/

/* Packet Driver Error numbers */
#define BAD_HANDLE      1     /* invalid handle number */
#define NO_CLAS         2     /* no interfaces of specified class found */
#define NO_TYPE         3     /* no interfaces of specified type found */
#define NO_NUMBER 4     /* no interfaces of specified number found */
#define BAD_TYPE  5     /* bad packet type specified */
#define NO_MULTICAST    6     /* this interface does not support multicast*/
#define CANT_TERMINATE  7     /* this packet driver cannot terminate */
#define BAD_MODE  8     /* an invalid receiver mode was specified */
#define NO_SPACE  9     /* failed because of insufficient space */
#define TYPE_INUSE      10    /* the type has already been accessed */
                        /* and not released. */
#define BAD_COMMAND     11    /* command out of range, or not implemented */
#define CANT_SEND 12    /* packet couldn't be sent (usually hardware) */
#define CANT_SET  13    /* hardware address couldn't be changed */
                        /* (more than 1 handle open) */
#define BAD_ADDRESS     14    /* hardware address has bad length or format */
#define CANT_RESET      15    /* couldn't reset interface */
                        /* (more than 1 handle open) */


#define RUNT            60    /* smallest legal size packet, no fcs */
#define GIANT           1514  /* largest legal size packet, no fcs */
#define EADDR_LEN 6     /* Ethernet address length. */


#define MAX_HANDLES 10        /* max number of handles at one time */
#define MIN_HANDLE 0          /* handles are 0 thru 9 */
#define MAX_TYPE_LEN   2      /* max packet type length */
#define OPEN  -1        /* available handle */

#define MIN(a,b)  (((a) < (b)) ? (a) : (b))
#define MAX(a,b)  (((a) > (b)) ? (a) : (b))

/* handle structure */
typedef struct _handle {
    int  in_use;        /* non-zero if handle exist */
    char type[MAX_TYPE_LEN];  /* packet type */
    int  len;                 /* packet length */
    unsigned int rec_es;      /* receiver address segment */
    unsigned int rec_di;      /* receiver address offset */
} HANDLE;

static unsigned char bit_swap[256] = {
    0x00, 0x80, 0x40, 0xc0, 0x20, 0xa0, 0x60, 0xe0,
    0x10, 0x90, 0x50, 0xd0, 0x30, 0xb0, 0x70, 0xf0,
    0x08, 0x88, 0x48, 0xc8, 0x28, 0xa8, 0x68, 0xe8,
    0x18, 0x98, 0x58, 0xd8, 0x38, 0xb8, 0x78, 0xf8,
    0x04, 0x84, 0x44, 0xc4, 0x24, 0xa4, 0x64, 0xe4,
    0x14, 0x94, 0x54, 0xd4, 0x34, 0xb4, 0x74, 0xf4,
    0x0c, 0x8c, 0x4c, 0xcc, 0x2c, 0xac, 0x6c, 0xec,
    0x1c, 0x9c, 0x5c, 0xdc, 0x3c, 0xbc, 0x7c, 0xfc,
```

TL/F/11142–22

```
    0x02, 0x82, 0x42, 0xc2, 0x22, 0xa2, 0x62, 0xe2,
    0x12, 0x92, 0x52, 0xd2, 0x32, 0xb2, 0x72, 0xf2,
    0x0a, 0x8a, 0x4a, 0xca, 0x2a, 0xaa, 0x6a, 0xea,
    0x1a, 0x9a, 0x5a, 0xda, 0x3a, 0xba, 0x7a, 0xfa,
    0x06, 0x86, 0x46, 0xc6, 0x26, 0xa6, 0x66, 0xe6,
    0x16, 0x96, 0x56, 0xd6, 0x36, 0xb6, 0x76, 0xf6,
    0x0e, 0x8e, 0x4e, 0xce, 0x2e, 0xae, 0x6e, 0xee,
    0x1e, 0x9e, 0x5e, 0xde, 0x3e, 0xbe, 0x7e, 0xfe,
    0x01, 0x81, 0x41, 0xc1, 0x21, 0xa1, 0x61, 0xe1,
    0x11, 0x91, 0x51, 0xd1, 0x31, 0xb1, 0x71, 0xf1,
    0x09, 0x89, 0x49, 0xc9, 0x29, 0xa9, 0x69, 0xe9,
    0x19, 0x99, 0x59, 0xd9, 0x39, 0xb9, 0x79, 0xf9,
    0x05, 0x85, 0x45, 0xc5, 0x25, 0xa5, 0x65, 0xe5,
    0x15, 0x95, 0x55, 0xd5, 0x35, 0xb5, 0x75, 0xf5,
    0x0d, 0x8d, 0x4d, 0xcd, 0x2d, 0xad, 0x6d, 0xed,
    0x1d, 0x9d, 0x5d, 0xdd, 0x3d, 0xbd, 0x7d, 0xfd,
    0x03, 0x83, 0x43, 0xc3, 0x23, 0xa3, 0x63, 0xe3,
    0x13, 0x93, 0x53, 0xd3, 0x33, 0xb3, 0x73, 0xf3,
    0x0b, 0x8b, 0x4b, 0xcb, 0x2b, 0xab, 0x6b, 0xeb,
    0x1b, 0x9b, 0x5b, 0xdb, 0x3b, 0xbb, 0x7b, 0xfb,
    0x07, 0x87, 0x47, 0xc7, 0x27, 0xa7, 0x67, 0xe7,
    0x17, 0x97, 0x57, 0xd7, 0x37, 0xb7, 0x77, 0xf7,
    0x0f, 0x8f, 0x4f, 0xcf, 0x2f, 0xaf, 0x6f, 0xef,
    0x1f, 0x9f, 0x5f, 0xdf, 0x3f, 0xbf, 0x7f, 0xff,
};
#define        BIT_SWAP(a)         bit_swap[(unsigned char )(a)]

#define        BYTE_SWAP(a, b)   { *(a) = *(b+1); *(a+1) = *(b); }

#define BUF_SZ 1514
static unsigned char s_buf[BUF_SZ];

static unsigned char snap[] =
    /* SNAP */
    { 170, 170, 3, 0, 0, 0 };

#define ETYPE_OFS 12
#define DATA_OFS 14
#define MAC_LEN 14

static struct {
    unsigned long  packets_in;
    unsigned long  packets_out;
    unsigned long  bytes_in;
    unsigned long  bytes_out;
    unsigned long  errors_in;
    unsigned long  errors_out;
    unsigned long  packets_dropped;
} drv_stats;
```

TL/F/11142–23

FILENAME: sonic.h

```
/* SONIC definition and data structures */

#define     iobase          0x300
#define     pagebase        0x30f
#define     regbase         0x310
#define     TDANUM          5
#define     RDANUM          40
#define     RRANUM          3
#define     RBA_BUF_SIZE    4096

/* isr bit pattern */
#define     CMD_LCAM        0x0200
#define     ISR_RFO         0x0001
#define     ISR_RBE         0x0020
#define     ISR_RDE         0x0040
#define     ISR_PKTRX       0x0400
#define     ISR_TXDN        0x0200
#define     ISR_TXER        0x0100
#define     ISR_LCD         0x1000

/****************************************************
 *                                       *
 * Offset of the register from the i/o base address *
 *                                       *
 ****************************************************/

#define cr      0       /* Command */
#define dcr     2       /* Data Configuration */
#define rcr     4       /* Receive Control */
#define tcr     6       /* Transmit Control */
#define imr     8       /* Interrupt Mask */
#define isr     10      /* Interrupt Status */
#define utda    12      /* Upper Transmit Descriptor Addr */
#define ctda    14      /* Current Transmit Descriptor Addr */
#define tps     0       /* Transmit Packet Size */
#define tfc     2       /* Transmit Fragment Count */
#define tsa0    4       /* Transmit Start Address 0 */
#define tsa1    6       /* Transmit Start Address 1 */
#define tfs     8       /* Transmit Fragment Size */
#define urda          10      /* Upper Receive Descriptor Addr */
#define crda          12      /* Current Receive Descriptor Addr */
#define crba0         14      /* Current Receive Buffer Addr 0 */
#define crba1         0       /* Current Receive Buffer Addr 1 */
#define rbwc0         2       /* Remaining Buffer Word Count 0 */
#define rbwc1         4       /* Remaining Buffer Word Count 1 */
#define eobc          6       /* End of Buffer Word Count */
#define urra          8       /* Upper Receive Resource Addr */
#define rsa           10      /* Resource Start Addr */
#define rea           12      /* Resource End Addr */
#define rrp           14      /* Resource Read Addr */
#define rwp           0       /* Resource Write Addr */
#define trba0         2       /* Temp Recv. Buffer Addr 0 */
#define trba1         4       /* Temp Recv. Buffer Addr 1 */
#define tbwc0         6       /* Temp Buffer Word Count 0 */
#define tbwc1         8       /* Temp Buffer Word Count 1 */
#define addr0         10      /* Address Generator 0 */
#define addr1         12      /* Address Generator 1 */
#define llfa          14      /* Last link Field Addr */
#define ttda          0       /* Temp Transmit Descriptor Addr */
```

TL/F/11142–24

```
#define cep            2       /* CAM entry Point */
#define cap2           4       /* CAM Address Port 2 */
#define cap1           6       /* CAM Address Port 1 */
#define cap0           8       /* CAM Address Port 0 */
#define ce             10      /* CAM Enable */
#define cdp            12      /* CAM Descriptor Pointer */
#define cdc            14      /* CAM Descriptor Count */
#define sr             0       /* Silicon Revision */
#define wt0            2       /* Watchdog Timer 0 */
#define wt1            4       /* Watchdog Timer 1 */
#define rsc            6       /* Receive Sequence Counter */
#define crct           8       /* CRC Error Tally */
#define faet           10      /* FAE Error Tally */
#define mpt      12      /* Missed Packet Tally */
#define mdt      14      /* Maximum Deferral Timer */
#define rtc      0       /* Receive Test Control */
#define ttc      2       /* Transmit Test Control */
#define dtc      4       /* DMA Test Control */
#define cc0      6       /* CAM Comparison 0 */
#define cc1      8       /* CAM Comparison 1 */
#define cc2      10      /* CAM Comparison 2 */
#define cm       12      /* CAM Match */
#define reserve1    14  /* Reserved */
#define reserve2    0   /* Reserved */
#define rbc      2       /* Receiver Byte Count */
#define reserve3    4   /* Reserved */
#define tbc      6       /* Transmitter Backoff Counter */
#define trc      8       /* Transmitter Random Counter */
#define tbm      10      /* Transmitter Backoff Mask */
#define reserve4    12  /* Reserved */
#define reserve5    14  /* Reserved */


/* tda structure */
typedef struct tda_construct {
      unsigned short    status;
      unsigned short    config;
      unsigned short    pkt_size;
      unsigned short    frag_count;
      unsigned short    frag_ptr0;
      unsigned short    frag_ptr1;
      unsigned short    frag_size;
      unsigned short    link;
}   tda_struct;

/* rda structure */
typedef struct rda_construct {
      unsigned short    status;
      unsigned short    byte_count;
      unsigned short    pkt_ptr0;
      unsigned short    pkt_ptr1;
      unsigned short    seq_no;
      unsigned short  pkt_link;
      unsigned short    in_use;
}   rda_struct;


/* rra structure */
typedef struct rra_construct {
        unsigned short  buff_ptr0;
        unsigned short  buff_ptr1;
        unsigned short  buff_wc0;
```

TL/F/11142–25

```
        unsigned short  buff_wc1;
}   rra_struct;

/* rba structure */
typedef struct rba_construct {
        unsigned char   buff[RBA_BUF_SIZE];
}   rba_struct;

typedef struct cam_port {
        unsigned short  entry_ptr;
        unsigned short  port0;
        unsigned short  port1;
        unsigned short  port2;
} cam_port_struct;

typedef struct cam_construct {
        cam_port_struct cam_port_info[16];
        unsigned short  cam_enable;
} cam_struct;

rba_struct   rba[RRANUM];
tda_struct   tda[TDANUM];
rda_struct   rda[RDANUM];
rra_struct   rra[RRANUM];
cam_struct   cam;

short transmitactive;               /* transmission currently active flag */
short curtda;                       /* current tda */
short currda;                       /* current rda */
short previous_seqno;               /* previous sequence number */
short retry;                        /* transmit retry counter */
unsigned short rwp_table[6];        /* RRA location table structure */
short cur_rwp;                      /* pointer to rwp_table */
unsigned short tda_addr;             /* tda starting address */
unsigned char far *type_ptr;        /* pointer for packet type */
```

TL/F/11142–26

FILENAME: isrlib.asm

```
;******************************************************************************
;*          Copyright (c) 1990 National Semiconductor Corporation            *
;*                              All Rights Reserved                           *
;******************************************************************************

_TEXT SEGMENT  WORD PUBLIC 'CODE'
_TEXT ENDS
_DATA SEGMENT  WORD PUBLIC 'DATA'
_DATA ENDS
CONST SEGMENT  WORD PUBLIC 'CONST'
CONST ENDS
_BSS  SEGMENT  WORD PUBLIC 'BSS'
_BSS  ENDS
DGROUP      GROUP CONST, _BSS, _DATA
      ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP

_TEXT segment     word public 'CODE'
      assume      cs:_TEXT

      public      _get_sp
_get_sp           proc far
      mov   ax,sp
      add   ax,4
      mov   dx,ss
      ret
_get_sp           ENDP

      public      _set_sp
_set_sp           proc far
      mov   bx,ss
      mov   es,bx
      mov   bx,sp

      pushf
      cli
      pop   dx

      mov   sp,word ptr ss:[bx+4]
      mov   ss,word ptr ss:[bx+6]

      and   dx,512
      jz    skip
      sti

skip: sub   sp,4
      mov   ax, word ptr es:[bx+2]
      push  ax
      mov   ax, word ptr es:[bx]
      push  ax
      ret
_set_sp           ENDP

      public      _get_if
_get_if           proc far
      pushf
      pop   dx
      mov   ax,0
      and   dx,512
      jz    ifret
```

TL/F/11142–27

```
      mov   ax,1
ifret:      ret
_get_if           ENDP

ARG_OFS     equ   6                 ;near = 4, far = 6 (from bp)
      public      _int_fddi
_int_fddi   proc far
      push  bp
      mov   bp, sp
      sub   sp, 8             ;work area for INT code

      ;put INT code on stack
      mov   byte ptr[bp - 2], 0cbh
      mov   ax, word ptr[bp + ARG_OFS]
      mov   [bp - 3], al
      mov   byte ptr[bp - 4], 0cdh
      mov   word ptr[bp - 6], ss
      lea   ax, word ptr[bp - 4]
      mov   word ptr[bp - 8], ax

      ;get regs values off sp, pointers are far
      push  bp
      mov   es, [bp + ARG_OFS + 4]
      mov   bp, [bp + ARG_OFS + 2]
      mov   ax, es:[bp]
      mov   bx, es:[bp + 2]
      mov   cx, es:[bp + 4]
      mov   dx, es:[bp + 6]
      mov   si, es:[bp + 8]
      mov   di, es:[bp + 10]
      pop   bp

      call  dword ptr[bp - 8] ;do INT

      ;get carry bit
      push  ax
      pushf
      pop   ax
      and   ax, 1             ;mask carry bit

      ;put regs values on sp
      mov   es, [bp + ARG_OFS + 8]
      mov   bp, [bp + ARG_OFS + 6]
      mov   es:[bp + 12], ax  ;cflag
      pop   ax
      mov   es:[bp], ax
      mov   es:[bp + 2], bx
      mov   es:[bp + 4], cx
      mov   es:[bp + 6], dx
      mov   es:[bp + 8], si
      mov   es:[bp + 10], di

      add   sp, 8
      pop   bp
      ret
_int_fddi   ENDP

_TEXT   ends
      end
```
TL/F/11142–28

FILENAME: pktint.asm

```
; ****************************************************************************
; *         Copyright (c) 1990 by National Semiconductor Corporation          *
; *                           All Rights Reserved                             *
; ****************************************************************************

      title TEXT - Interrupt service routine

      extrn _int_handler:near

_TEXT SEGMENT  WORD PUBLIC 'CODE'
_TEXT ENDS
_DATA SEGMENT  WORD PUBLIC 'DATA'
_DATA ENDS
CONST SEGMENT  WORD PUBLIC 'CONST'
CONST ENDS
_BSS  SEGMENT  WORD PUBLIC 'BSS'
_BSS  ENDS
DGROUP      GROUP CONST, _BSS, _DATA
      ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP

_DATA SEGMENT     WORD PUBLIC 'DATA'
      assume      ds:DGROUP
rcvr_ptr    dd    ?
segmoffs    struc
offs        dw    ?
segm        dw    ?
segmoffs    ends

_DATA ENDS


_TEXT segment     word public 'CODE'
      assume      cs:_TEXT

CFLAG_OFFSET      equ   2
FLAG_OFFSET equ   6
REGS_OFFSET equ   14
SREGS_OFFSET      equ   22


      public _drv_isr
_drv_isr    proc  far

      jmp   start
      db    'PKT DRVR',0            ;driver signature

;setup registers on stack for MSC's union REGS and struct SREGS
start:


      assume      ds:nothing

      push  bp
      mov   bp, sp
      and   word ptr[bp+FLAG_OFFSET], not 1     ;clear carry bit
      push  word ptr[bp+FLAG_OFFSET] ;put in cflag field of structure
      push  di                ;save regular registers
      push  si
      push  dx
```

TL/F/11142–29

```
      push  cx
      push  bx
      push  ax
      push  ds                ;save segment registers
      push  ss
      push  cs
      push  es

      push  ss
      lea   ax, word ptr [bp-SREGS_OFFSET]      ;pass sregs pointer
      push  ax
      push  ss
      lea   ax, word ptr [bp-REGS_OFFSET] ;pass regs pointer -> ax
      push  ax

      mov   ax, DGROUP              ;get global data segment
      mov   ds, ax                        ;make segment addressable
      assume ds: DGROUP
      cld
      call  _int_handler                  ;call C interrupt handler
      add   sp, 8

      mov   ax, word ptr[bp-CFLAG_OFFSET] ;mov cflag to flag reg
      mov   word ptr[bp+FLAG_OFFSET], ax

      pop   es                      ;restore registers
      pop   ax                      ;dummy pop for cs
      pop   ss
      pop   ds
      pop   ax
      pop   bx
      pop   cx
      pop   dx
      pop   si
      pop   di
      pop   bp                      ;pop cflag of structure
      pop   bp

      iret                          ;return from interrupt
_drv_isr    endp


      public _app_recv
_app_recv  proc near
ax_ofs      equ   4

      assume  ds:DGROUP
      push  bp
      mov   bp, sp
      push  ds
      push  es
      push  bx

      mov   bx, [bp+ax_ofs+10]      ;set-up app reciever
      mov   rcvr_ptr.offs, bx
      mov   bx, [bp+ax_ofs+12]
      mov   rcvr_ptr.segm, bx

      les   bx, dword ptr[bp+ax_ofs+6] ;buffer
      mov   si, word ptr es:[bx]
      push  ds
      mov   ds, word ptr es:[bx+2]
```

TL/F/11142–30

```
        mov    ax, [bp+ax_ofs]
        mov    bx, [bp+ax_ofs+2]
        mov    cx, [bp+ax_ofs+4]
        pop    es
        assume  es:DGROUP

        call   es:rcvr_ptr

        mov    ax, es
        les    bx, dword ptr[bp+ax_ofs+6]    ;update pointer ES:DI
        mov    word ptr es:[bx], di
        mov    word ptr es:[bx+2], ax
        pop    bx
        pop    es
        pop    ds
        pop    bp
        ret                            ;return

_app_recv   endp

_TEXT   ends
        end
```

TL/F/11142–31

FILENAME: pktdrv.mak

```
ZI     =
INC    = ..\include
CFLAGS       = $(ZI) -Gs -c -I$(INC)
MFLAGS       = -Ml

OBJ    = pktdrv.obj sonic.obj pktint.obj far.obj isr.obj isrlib.obj
#LIB   = ..\lib\frame.lib
LIB     =

sonic.obj:  sonic.c $(INC)\sonic.h
    cl $(CFLAGS) $*.c

pktdrv.obj: pktdrv.c $(INC)\pktdrv.h $(INC)\sonic.h
    cl $(CFLAGS) $*.c

far.obj: far.c $(INC)\sonic.h
    cl $(CFLAGS) $*.c

isr.obj: isr.c $(INC)\sonic.h
    cl $(CFLAGS) $*.c

isrlib.obj: isrlib.asm
    masm $(MFLAGS) $*.asm;

pktint.obj: pktint.asm
    masm $(MFLAGS) $*.asm;

pktdrv.exe: $(OBJ)
    cl $(ZI) $(OBJ)  -o $*

#pktdrv.exe: $(OBJ)
#    link /CO /LI /MAP $(OBJ), $*, ,$(LIB);
#    msym pktdrv
```

TL/F/11142–32

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco 120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998