DP83932EB-EISA SONIC/ EISA Packet Driver for PC/TCP

National Semiconductor
Application Note 859
August 1992

INTRODUCTION

This is a complete program listing for a network device driver for the DP83932EB-EISA SONIC EISA Demonstration and Evaluation Board. This driver enables the DP83932-EISA to operate with the Personal Computer-based TCP/IP software package, distributed by FTP Software Inc., called PC/TCP. Contact FTP Software Inc. at (617) 246-0900 for more information about the PC/TCP product offerings.

This driver conforms to version 1.9 of FTP Software's Packet Driver Specification, and works with verison 2.x of the PC/TCP product (and products that have adopted the Packet Driver Specification).

This program listing is provided as an example of drive software for the DP83932 Systems Oriented Network Interface Controller (SONIC). The driver is written in Microsoft C (5.1 or greater) and Microsoft Assembler (5.1 or greater). Since the majority of the software is written in C, the concepts provided are easily portable to other environments.

This example software is provided as an example, and is not necessarily the most optimal implementation. The code has been thoroughly tested with PC/TCP.

The driver is listed by modules in the following order:

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. makefile

©1995 National Semiconductor Corporation RRD-B30M75/Printed in U. S. A.

PKTDRV.C

```
static char pktdrv_rcsid[]="@(#)$ID:$";
/*
*****************************************************************************
*         Copyright (c) 1992 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.
============================================================================
UPDATE LOG:
When/Who                        Why/What/Where
----------------------------------------------------------------------------
11/30/90 Mike Lui               Convert to work for SONIC 32 bit
04/10/92 Michael Zhang          Added read_config();
                                Added 'transmitactive=1' in send_packet();

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

#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 */

  /* 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 = 5;          /* basic and high-
                                                   performance driver function */
static char drv_name[] =                /* driver name */
    "National Semiconductor SONIC/TCP 32-bit Packet Driver";
static char cpy_msg[] =
    "Copyright (c) 1992 by National Semiconductor Corporation";
static char drv_rev[] = "1.2";          /* current driver rev */
static unsigned char BOARD_ID[]={ 0x41,0x98,0x10,0x01 }; /* PLX1001 */
static HANDLE handle_tbl[MAX_HANDLES];  /* create handle structs */
static void read_config();              /* read board config info */
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 */
unsigned char type_buf[MAX_TYPE_LEN];
static void usage();

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

TL/F/11720–1

```
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);

    read_config();              /* read expansion board config*/

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

    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.
 *
 *      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;
```

TL/F/11720–2

```
    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 10:
        ret_val = get_param(regs, sregs);      /* high-performance function */
        break;
    case 11:
        ret_val = as_send_pkt(regs, sregs);    /* high-performance function */
        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 */
    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.
```

TL/F/11720–3

```
 *
 *      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;

    /* 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 */
        }
        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++;
```

TL/F/11720–4

```
    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;
    tda_struct *tmp_tda;
    short previous_tda;
    unsigned short addr;

    /* check if frame is too big */
    if(regs->x.cx > BUF_SZ) {
        return NO_SPACE;
    }

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

TL/F/11720–5

```
    /* 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)
            return CANT_SEND;
        else {
        /* copy data area from the frame */
            far_memcpy((char far *)&tba[curtda+1], &frame_ptr[0], regs->x.cx);
            addr=tda_addr+(curtda+1)*sizeof(tda_struct);
            tmp_tda=(tda_struct*)addr;
            tmp_tda->pkt_size=buf_len;
            tmp_tda->frag_count=1;
            tmp_tda->frag_size=buf_len;
            tmp_tda->link |= 1;
            tmp_tda->type = BASIC;
            addr-=sizeof(tda_struct);
            tmp_tda=(tda_struct*)addr;
            tmp_tda->link &= 0x0fffe;
            curtda++;
        }
    }
    else {
        /* network is free */
        retry=0;
        /* copy data area from the frame */
        far_memcpy((char far *)&tba[0], &frame_ptr[0], regs->x.cx);
        tmp_tda=(tda_struct*) tda_addr;
        tmp_tda->pkt_size=buf_len;
        tmp_tda->frag_count=1;
        tmp_tda->frag_size=buf_len;
        tmp_tda->link |= 1;
        tmp_tda->type = BASIC;
        tda_head=0;
        tda_tail=1;
        curtda=0;
        outpw(regbase+ctda, tda_start_addr);  /* load ctda */
        transmitactive=1;
    }

    outpw(regbase+cr, 2);           /* issue the transmit command */

    return 0;
}

/*
 * terminate()
 *
 *  Terminate the driver.
 *
 *      Return values:   0 - Success
 *                      >0 - Failure
```

TL/F/11720–6

```
 */
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;
{
    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;

    regs->x.cx = 6;

    for(i = 0; i < regs->x.cx; i++) {
        addr_ptr[i] = inp(regbase+0xc90+i);
    }

    return 0;
}
```

TL/F/11720–7

```
/*
 * 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_param()
 *
 *  Return driver parameters
 *
 *      Return values:   0 - Success
 *                      >0 - Failure
 */
get_param(regs, sregs)
union REGS far *regs;
struct SREGS far *sregs;
{
    if(drv_funct !=5 && drv_funct != 6)
        return BAD_COMMAND;

    drv_param.major_rev=1;
    drv_param.minor_rev=9;
    drv_param.length=14;
    drv_param.addr_len=6;
    drv_param.mtu=1512;
    drv_param.multicast_aval=90;
    drv_param.rcv_bufs=3;
    drv_param.xmt_bufs=3;
    drv_param.int_num=0;
```

TL/F/11720–8

```
    regs->x.di = (unsigned)&drv_param;          /* driver stats */
    sregs->es  = (unsigned long)((char far *)&drv_param) >> 16;

    return 0;
}


/*
 * as_send_pkt()
 *
 *  HIgh performance send packet.
 *
 *      Return values:   0 - Success
 *                      >0 - Failure
 */
as_send_pkt(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;
    tda_struct *tmp_tda;
    short previous_tda;
    unsigned short addr;

    /* check if frame is too big */
    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) {
            xmt_upcall(CANT_SEND, (char far *) &frame_ptr,regs->x.di,sregs->es);
            return CANT_SEND;
        }
        else {
        /* copy data area from the frame */
            far_memcpy((char far *)&tba[curtda+1], &frame_ptr[0], regs->x.cx);
            addr=tda_addr+(curtda+1)*sizeof(tda_struct);
            tmp_tda=(tda_struct*)addr;
            tmp_tda->pkt_size=buf_len;
            tmp_tda->frag_count=1;
```

TL/F/11720–9

```
            tmp_tda->frag_size=buf_len;
            tmp_tda->link |= 1;
            tmp_tda->type = HIGH_PERFORMANCE;
            tmp_tda->buffer=frame_ptr;
            tmp_tda->xmt_es=sregs->es;
            tmp_tda->xmt_di=regs->x.di;
            addr-=sizeof(tda_struct);
            tmp_tda=(tda_struct*)addr;
            tmp_tda->link &= 0x0fffe;
            curtda++;
            tda_tail=curtda+1;
        }
    }
    else {
        /* network is free */
        retry=0;
        /* copy data area from the frame */
        far_memcpy((char far *)&tba[0], &frame_ptr[0], regs->x.cx);
        tmp_tda=(tda_struct*) tda_addr;
        tmp_tda->pkt_size=buf_len;
        tmp_tda->frag_count=1;
        tmp_tda->frag_size=buf_len;
        tmp_tda->link |= 1;
        tmp_tda->type = HIGH_PERFORMANCE;
        tmp_tda->buffer=frame_ptr;
        tmp_tda->xmt_es=sregs->es;
        tmp_tda->xmt_di=regs->x.di;
        curtda=0;
        tda_head=0;
        tda_tail=1;
        outpw(regbase+ctda, tda_start_addr);  /* load ctda */
    }

    outpw(regbase+cr, 2);           /* issue the transmit command */

    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;

    return 0;
}
```

TL/F/11720–10

```
/*
 * 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 */
    char far *cp_ptr;
    unsigned short addr;
    unsigned char far *frame;


    /* get the frame */
    while ((unsigned short)cur_rda->status != 0) {
        frame=(unsigned char far *) (((unsigned long) cur_rda->pkt_ptr1 << 28) |
                                     (unsigned short) cur_rda->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) == 0))
                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;
        }
        if ((unsigned short) cur_rda->status & 0x0c) {
            drv_stats.packets_dropped++;
            free_rda();
            continue;
        }

        /* update driver stats */
        drv_stats.packets_in++;
        drv_stats.bytes_in += (unsigned short) cur_rda->byte_count;

        /* first upcall, tell them frame size */
        app_recv(0,handle_found, MAX((unsigned short) cur_rda->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 */
```

TL/F/11720–11

```
        if(cp_ptr == NULL) {
            drv_stats.packets_dropped++;
            free_rda();
            continue;
        }

        /* copy the frame */
        far_memcpy(&cp_ptr[0], &frame[0], (unsigned short)cur_rda->byte_count-4)

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

        /* free rda */
        free_rda();
    }
    return 0;
}



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

free_rda()
{
    static int first;
    unsigned short tmp_value;
    unsigned short addr;
    rda_struct * p_rda;


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

    /* reinitialize the rda */
    cur_rda->status=0;
    cur_rda->byte_count=0;
    cur_rda->pkt_ptr0=0;
    cur_rda->pkt_ptr1=0;
    cur_rda->in_use=0x0ffff;
    cur_rda->pkt_link |= 1;

    /* link the previous rda to the current rda */
    if (currda==0) {
        addr=rda_start_addr+(RDANUM-1)*sizeof(rda_struct);
        p_rda=(rda_struct*) addr;
        p_rda->pkt_link&=0x0fffe;
    }
    else {
        addr=c_rda-sizeof(rda_struct);
        p_rda=(rda_struct*) addr;
        p_rda->pkt_link&=0x0fffe;
```

TL/F/11720–12

```
    }

    /* get the first buffer number */
    if (!first) {
        previous_seqno=(unsigned short)cur_rda->seq_no >> 8;
        first=1;
    }

    /* check whether rba can be reused */
    if ((unsigned short)cur_rda->seq_no >> 8 != previous_seqno) {
        previous_seqno=(unsigned short)cur_rda->seq_no >> 8;
        tmp_value=rwp_table[cur_rwp];
        if (cur_rwp==2)
            cur_rwp=0;
        else
            cur_rwp++;

        outpw(regbase + rwp, tmp_value);

        tmp_value=inpw(regbase + isr);
        if (tmp_value & ISR_RBE)
            outpw(regbase + isr, ISR_RBE);
    }

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

    if (currda == RDANUM-1) {
        currda=0;
        c_rda=rda_start_addr;
        cur_rda=(rda_struct*)c_rda;
    }
    else {
        currda++;
        c_rda+=sizeof(rda_struct);
        cur_rda=(rda_struct*)c_rda;
    }

}


/*
 * 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);
```

TL/F/11720–13

```
    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);
}

/*
 * 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",
```

TL/F/11720–14

```
                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)
int  argc;
char *argv[];
{
    int in, done = 0,c_type;
    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 (((in = getopt(argc, argv, "?khi:t:")) != -1)) {
        switch(in) {
        case 'k':
                return (1);
            break;
        case 't':
            sscanf(optarg, "%d", &c_type);
            if(c_type==1) cable_type=THICK;
            break;
        case 'i':
            if(sscanf(optarg, "0x%x", &packet_int_no) != 1)
                if(sscanf(optarg, "%d", &packet_int_no) != 1) {
                    break;
                }
            /*
            if(!strncmp(optarg, "0x", 2))
                sscanf(&optarg[2], "%x", &packet_int_no);
            else
                sscanf(optarg, "%d", &packet_int_no);
```

TL/F/11720–15

```
            */
            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);
            }
            break;
        default:
            usage(argv);
            break;
        }
    }
}

void usage(argv)
char **argv;
{
            fprintf(stderr,
             "Usage: %s [-h] [-k] [-i packet_int_no] [-t cable type]\n", *argv);
            fprintf(stderr, "  -h = this help message\n");
            fprintf(stderr,
                "  -i = set packet interrupt number, default is 0x60\n");
            fprintf(stderr, "  -t = cable type (0 thin coax, 1 AUI)\n");
            fprintf(stderr, "  -k = 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.
 *             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)
```

TL/F/11720–16

```
      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() */

void read_config()
{
        unsigned short reg0,i;
        unsigned short port;

        for(i=0; i<MAX_SLOT; i++)       /* read board ID */
        {
                port=(0x1000)*i + ID_ADDR;
                if(inpw(port)==*(unsigned int *)BOARD_ID &&
                   inpw(port+2)==*(unsigned int *)(BOARD_ID+2)
                )
                        break;
        }
        if( i==MAX_SLOT ) {     /* no board found */
                fprintf(stderr,"No PLX board found.\n");
                exit (1);
        }

        regbase=0x1000 * i;

        reg0=inp(regbase+0xc88);        /* read plx register 0 */
        reg0 &=0x05;                    /* bit 2,1 */

        switch (reg0) {
                case 0: sonic_irq=5;
                        break;
                case 2: sonic_irq=9;
                        break;
                case 4: sonic_irq=10;
                        break;
                case 6: sonic_irq=11;
                        break;
        }

        reg0=inp(regbase+0xc89);        /* read plx register 1 */
        if( reg0 & 0x02 ) cable_type=THIN;
        else cable_type=THICK;
}
```

TL/F/11720–17

FAR.C

```
static char far_rcsid[]="@(#)$ID:$";
/*
 ****************************************************************************
 *           Copyright (c) 1992 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/11720–18

ISR.C

```
 static char isr_csid[]="@(#)$ID:$";
/*
 ********************************************************************************
 *           Copyright (c) 1992 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%8));
    _enable();

    if(irq>8) {         /* also enable PIC 1 */
        int tmp_mask;
        int tmp_pic_ctl;
        int tmp_pic_mask;

        tmp_pic_ctl=0x20;
        tmp_pic_mask = tmp_pic_ctl +1;
        tmp_mask=inp(tmp_pic_mask);
        _disable();
        outp(tmp_pic_mask,tmp_mask & ~(1 << 2));
        _enable();
    }
}


void sonic_isr_disable(irq)
int irq;
{
    _disable();
    _dos_setvect(irq_map[irq], sys_irq_int);
```

TL/F/11720–19

```
    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 activetda, addr;
    unsigned short isr_reg;
    short i;
    tda_struct * tmp_tda;

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

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

    _enable();

    isr_reg=inpw(regbase+isr);

    while (isr_reg) {
        if (isr_reg & ISR_PKTRX)  {         /* is there a receive */
            outpw(regbase+isr, ISR_PKTRX);  /* clear receive bit */
            drv_rcvr();                     /* process rda */
        }
        if (isr_reg & ISR_TXDN) {           /* is there is transmit done */
            outpw(regbase+isr, ISR_TXDN);
            transmitactive=0;
            for (i=tda_head; i<tda_tail; i++) {
                addr=tda_addr+i*sizeof(tda_struct);
                tmp_tda=(tda_struct *) addr;
                if ((unsigned short)tmp_tda->type==HIGH_PERFORMANCE)
                    xmt_upcall(0, (char far *) &tmp_tda->buffer,
                       (unsigned short)tmp_tda->xmt_di, (unsigned short)tmp_tda-
            }
        }
        if (isr_reg & ISR_TXER) {           /* is there a transmit error */
            outpw(regbase+isr, ISR_TXER);
            if (retry > 10) {    /* if retry 10 and still not succeed to transmi
                activetda=inpw(regbase+ctda);
                if (activetda & 0x1)
                    transmitactive=0;
                else {
                activetda &= 0x0fffe;
                outpw(regbase+ctda, activetda+20);
                outp(regbase+cr, 2);        /* transmit */
                }
            }
            else {                          /* try again */
                retry++;
                outp(regbase+cr, 2);        /* transmit */
            }
        }
        if (isr_reg & 0x0020)
```

TL/F/11720–20

```
            drv_rcvr();                    /* process rda */

        isr_reg=inpw(regbase+isr);
        isr_reg &=0x0700;
    }

    _disable();

    set_sp(old_sp);

    if(pic_ctl== 0xa0) outp(0x20,0x20);
    outp(pic_ctl, 0x20);

    outpw(regbase+imr, 0x0700);

}
```

TL/F/11720–21

SONIC.C

```
static char sonic_rcsid[]="@(#)$ID:$";
/*
 ************************************************************************
 *     Copyright (c) 1992 by National Semiconductor Corporation         *
 *                        All Rights Reserved                           *
 ************************************************************************
 *
 */

#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;

    /* initialize valuables */
    transmitactive=0;
    curtda=0;
    currda=0;

    /* initialize the EISA9010 chip */

        /* register 1 */
    /*cable_type=THICK;*/
    outpw(regbase+plx_reg1,cable_type);

    /* install sonic interrupt */
    sonic_isr_enable(sonic_irq);

    /* initialize sonic register */
    outpw(regbase+cr, 0x94);      /* reset sonic */
    outpw(regbase+dcr, 0x073a);   /* set configuration: 0 wait state
                                                        32-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);   /* accept broadcast packet */
    outpw(regbase+isr, 0x0ffff);  /* reset isr */
    outpw(regbase+imr, 0x0700);    /* 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 */
```

TL/F/11720–22

```
    /* initialize rwp location table */
    cur_loc=inpw(regbase+rsa);
    for (i=0; i<RRANUM; i++) {
        rwp_table[i]=cur_loc;
        cur_loc+=16;
    }
    cur_rwp=0;

    /* normal operation */
    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;
    unsigned long tba_addr;
    char far *ptr;
    struct SREGS segregs;
    tda_struct *tmp_tda;
    unsigned short c_tda_addr;
    unsigned short n_tda_addr;

    segread(&segregs);      /* Read the segment register value */
    /* check double word boundry */
    tda_addr=(unsigned short) &tda[0];
    tda_addr&=0xfffc;
    /* link the first nine tda */
    for (i=0; i<TDANUM-1; i++) {
      c_tda_addr=tda_addr+i*sizeof(tda_struct);
      n_tda_addr=c_tda_addr+sizeof(tda_struct);
      addr32=(((unsigned long) segregs.ds << 16) | n_tda_addr);
      tba_addr=(((unsigned long)segregs.ds << 16) |
               ((unsigned short) &tba[i]));
      u16=addr32>>16;
      l16=(unsigned short)addr32;
      addr32=(unsigned long)u16 * 16 + l16;
      tmp_tda=(tda_struct*) c_tda_addr;
      tmp_tda->config=0x1000;
      tmp_tda->link=(unsigned short) addr32;
      u16=tba_addr>>16;
      l16=(unsigned short)tba_addr;
      tba_addr=(unsigned long)u16 * 16 + l16;
      tmp_tda->frag_ptr1=tba_addr>>16;
      tmp_tda->frag_ptr0=(unsigned short) tba_addr;
    }
```

TL/F/11720–23

```
    /* set the last tda link field to the first tda */
    addr32=(((unsigned long) segregs.ds << 16) | tda_addr);
    tba_addr=(((unsigned long) segregs.ds << 16) |
             ((unsigned short) &tba[TDANUM-1]));
    u16=addr32>>16;
    l16=(unsigned short)addr32;
    addr32=(unsigned long)u16 * 16 + l16;
    c_tda_addr=tda_addr+(TDANUM-1)*sizeof(tda_struct);
    tmp_tda=(tda_struct*) c_tda_addr;
    tmp_tda->link=(unsigned short) addr32;
    u16=tba_addr>>16;
    l16=(unsigned short)tba_addr;
    tba_addr=(unsigned long)u16 * 16 + l16;
    tmp_tda->frag_ptr1=tba_addr>>16;
    tmp_tda->frag_ptr0=(unsigned short) tba_addr;

    /* set the utda and ctda register */
    outpw(regbase+utda, addr32>>16);      /* set utda */
    outpw(regbase+ctda, (unsigned short)addr32);          /* set ctda */
    tda_start_addr=(unsigned short)addr32;
}




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

init_rda()
{
    unsigned short i, u16, l16;
    unsigned long addr32;
    struct SREGS segregs;
    rda_struct *tmp_rda;
    unsigned short c_rda_addr;
    unsigned short n_rda_addr;

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

    /* check double word boundry */
    rda_addr=(unsigned short) &rda[0];
    rda_addr&=0xfffc;
    c_rda=rda_addr;
    rda_start_addr=c_rda;
    cur_rda=(rda_struct *) c_rda;
    /* link the rda */
    for (i=0; i<RDANUM-1; i++) {
        c_rda_addr=rda_addr+i*sizeof(rda_struct);
        n_rda_addr=c_rda_addr+sizeof(rda_struct);
        addr32=(((unsigned long) segregs.ds << 16) | n_rda_addr);
        u16=addr32>>16;
        l16=(unsigned short)addr32;
        addr32=(unsigned long)u16 * 16 + l16;
```

TL/F/11720–24

```
        tmp_rda=(rda_struct*) c_rda_addr;
        tmp_rda->pkt_link=(unsigned short) addr32;
        tmp_rda->in_use=0x0ffff;
    }

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

    /* set the urda and crda register */
    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;
    unsigned short rra_addr, addr;
    rra_struct * tmp_rra;

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

    /* check double word boundry */
    rra_addr=(unsigned short) &rra[0];
    rra_addr&=0xfffc;
    /* 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;
        addr=rra_addr+i*sizeof(rra_struct);
        tmp_rra=(rra_struct*) addr;
        tmp_rra->buff_ptr0=(unsigned short)addr32;
        tmp_rra->buff_ptr1=addr32>>16;
        tmp_rra->buff_wc0=RBA_BUF_SIZE/2;
        tmp_rra->buff_wc1=0;
    }

    addr32=(((unsigned long) segregs.ds << 16) | rra_addr);
```

TL/F/11720–25

```
    u16=addr32>>16;
    l16=(unsigned short)addr32;
    addr32=(unsigned long)u16 * 16 + l16;

    /* set urra, rsa, and rrp */
    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+=48;
    outpw(regbase+rea, (unsigned short) addr32);         /* set rea */
    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;
    unsigned short cam_addr, addr;
    cam_struct * tmp_cam;

    segread(&segregs);      /* Read the segment register value */
    /* check double word boundry */
    cam_addr=(unsigned short) &cam[0];
    cam_addr&=0xfffc;

    addr32=(((unsigned long) segregs.ds << 16) | cam_addr);
    u16=addr32>>16;
    l16=(unsigned short)addr32;
    addr32=(unsigned long)u16 * 16 + l16;

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

    tmp_cam=(cam_struct *) cam_addr;
    /* load the cda with node physical address */
    tmp_cam->cam_port_info[0].port0=inpw(regbase+0x0c90);
    tmp_cam->cam_port_info[0].port1=inpw(regbase+0x0c92);
    tmp_cam->cam_port_info[0].port2=inpw(regbase+0x0c94);

    for(i=0; i<16; i++)
        tmp_cam->cam_port_info[i].entry_ptr=i;

    tmp_cam->cam_enable=1;                           /* load cam enable */

    /* load cam */
    outpw(regbase+cr, CMD_LCAM);
```

TL/F/11720–26

```
    /* 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/11720–27

PKTDRV.H

```
/*
* $ID:$
*
*****************************************************************************
*         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,
```

TL/F/11720–28

```
    0x1c, 0x9c, 0x5c, 0xdc, 0x3c, 0xbc, 0x7c, 0xfc,
    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;

static struct {
    unsigned char  major_rev;
    unsigned char  minor_rev;
    unsigned char  length;
    unsigned char  addr_len;
    unsigned short mtu;
    unsigned short multicast_aval;
    unsigned short rcv_bufs;
    unsigned short xmt_bufs;
```

TL/F/11720–29

```
    unsigned short int_num;
} drv_param;
```

TL/F/11720–30

SONIC.H

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

/* SONIC definition and data structures */

#define     TDANUM          5
#define     RDANUM          40
#define     RRANUM          3
#define     RBA_BUF_SIZE    8192
#define     TBA_BUF_SIZE    1514

/* 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

#define     THIN            0x03
#define     THICK           0x01
#define     ID_ADDR         0xC80
#define     MAX_SLOT        15

/*********************************************************
 *                                                       *
 * Offset of the EISA9010 register from the regbase address *
 *                                                       *
 *********************************************************/
#define plx_ebc     0xC84       /* EBC register */
#define plx_reg0    0xC88       /* register 0 */
#define plx_reg1    0xC89       /* register 1 */
#define plx_reg2    0xC8A       /* register 2 */
#define plx_reg3    0xC8F       /* register 3 */
/*************************************************
 *                                                *
 * 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         16  /* Transmit Packet Size */
#define tfc         18  /* Transmit Fragment Count */
#define tsa0        20  /* Transmit Start Address 0 */
```

TL/F/11720–31

```
#define tsa1         22  /* Transmit Start Address 1 */
#define tfs          24  /* Transmit Fragment Size */
#define urda         26  /* Upper Receive Descriptor Addr */
#define crda         28  /* Current Receive Descriptor Addr */
#define crba0        30  /* Current Receive Buffer Addr 0 */
#define crba1        32  /* Current Receive Buffer Addr 1 */
#define rbwc0        34  /* Remaining Buffer Word Count 0 */
#define rbwc1        36  /* Remaining Buffer Word Count 1 */
#define eobc         38  /* End of Buffer Word Count */
#define urra         40  /* Upper Receive Resource Addr */
#define rsa          42  /* Resource Start Addr */
#define rea          44  /* Resource End Addr */
#define rrp          46  /* Resource Read Addr */
#define rwp          48  /* Resource Write Addr */
#define trba0        50  /* Temp Recv. Buffer Addr 0 */
#define trba1        52  /* Temp Recv. Buffer Addr 1 */
#define tbwc0        54  /* Temp Buffer Word Count 0 */
#define tbwc1        56  /* Temp Buffer Word Count 1 */
#define addr0        58  /* Address Generator 0 */
#define addr1        60  /* Address Generator 1 */
#define llfa         62  /* Last link Field Addr */
#define ttda         64  /* Temp Transmit Descriptor Addr */
#define cep          66  /* CAM entry Point */
#define cap2         68  /* CAM Address Port 2 */
#define cap1         70  /* CAM Address Port 1 */
#define cap0         72  /* CAM Address Port 0 */
#define ce           74  /* CAM Enable */
#define cdp          76  /* CAM Descriptor Pointer */
#define cdc          78  /* CAM Descriptor Count */
#define sr           80  /* Silicon Revision */
#define wt0          82  /* Watchdog Timer 0 */
#define wt1          84  /* Watchdog Timer 1 */
#define rsc          86  /* Receive Sequence Counter */
#define crct         88  /* CRC Error Tally */
#define faet         90  /* FAE Error Tally */
#define mpt          92  /* Missed Packet Tally */
#define mdt          94  /* Maximum Deferral Timer */
#define rtc          96  /* Receive Test Control */
#define ttc          98  /* Transmit Test Control */
#define dtc         100 /* DMA Test Control */
#define cc0         102 /* CAM Comparison 0 */
#define cc1         104 /* CAM Comparison 1 */
#define cc2         106 /* CAM Comparison 2 */
#define cm          108 /* CAM Match */
#define reserve1    110 /* Reserved */
#define reserve2    112 /* Reserved */
#define rbc         114 /* Receiver Byte Count */
#define reserve3    116 /* Reserved */
#define tbc         118 /* Transmitter Backoff Counter */
#define trc         120 /* Transmitter Random Counter */
#define tbm         124 /* Transmitter Backoff Mask */
#define reserve4    126 /* Reserved */
#define reserve5    128 /* Reserved */

#define BASIC               0
#define HIGH_PERFORMANCE    1

/* tda structure */
typedef struct tda_construct {
        unsigned long   status;
```

TL/F/11720–32

```
        unsigned long   config;
        unsigned long   pkt_size;
        unsigned long   frag_count;
        unsigned long   frag_ptr0;
        unsigned long   frag_ptr1;
        unsigned long   frag_size;
        unsigned long   link;
        unsigned long   type;
        char far *      buffer;
        unsigned long   xmt_di;
        unsigned long   xmt_es;
}   tda_struct;

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


/* rra structure */
typedef struct rra_construct {
        unsigned long  buff_ptr0;
        unsigned long  buff_ptr1;
        unsigned long  buff_wc0;
        unsigned long  buff_wc1;
}   rra_struct;

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

/* tba structure */
typedef struct tba_construct {
        unsigned char tba_buff[TBA_BUF_SIZE];
} tba_struct;

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

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

rba_struct   rba[RRANUM];
tba_struct   tba[TDANUM];
unsigned char tda[TDANUM*sizeof(tda_struct)+3];
unsigned char rda[RDANUM*sizeof(rda_struct)+3];
unsigned short in_isr;
```

TL/F/11720–33

```
unsigned char rra[RRANUM*sizeof(rda_struct)+3];
unsigned char cam[sizeof(cam_struct)+3];

unsigned short sonic_irq;          /* sonic interrupt*/
unsigned short cable_type;         /* thin/thick cable */
unsigned short regbase;            /* base io address */
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 short tda_start_addr;     /* tda starting physical address */
unsigned short rda_addr;           /* rda starting address */
unsigned short c_rda;
unsigned short rda_start_addr;
unsigned char far *type_ptr;       /* pointer for packet type */
short tda_head;                    /* head ptr to tda list */
short tda_tail;                    /* tail ptr to tda list */
rda_struct * cur_rda;
```

TL/F/11720–34

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
```

TL/F/11720–35

```
        jz      ifret
        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/11720–36

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      ?
upcall_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
```

TL/F/11720–37

```
        push    si
        push    dx
        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
```

TL/F/11720–38

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


        public _xmt_upcall
_xmt_upcall  proc near
ret_ofs equ   4

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

        mov     bx, [bp+ret_ofs+6]
        mov     upcall_ptr.offs, bx
        mov     bx, [bp+ret_ofs+8]
        mov     upcall_ptr.segm, bx

        les     bx, dword ptr[bp+ret_ofs+2] ;buffer
        mov     di, word ptr ds:[bx]
        mov     es, word ptr ds:[bx+2]
        mov     ax, [bp+ret_ofs]
        assume  ds:DGROUP

        call    ds:upcall_ptr

        pop     bx
        pop     es
        pop     ds
        pop     bp
        ret                                     ;return

_xmt_upcall     endp


_TEXT   ends
        end
```

TL/F/11720–39

MAKEFILE

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

OBJ     = pktdrv.obj sonic.obj pktint.obj far.obj isr.obj isrlib.obj
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 $*

clean:
        -del *.obj
```

TL/F/11720–40

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

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

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.