```
/*
**      header for VMEexec environment LANCE driver
**      %Z%%Y%%M%       %I%     %E%
*/

#ident  "%Z%%Y%%M%   %I%     %E%"

#define INKERNEL
#include <sys/types.h>
#include <sys/mvme147.h>
#include <vmex/rteid.h>
#include <vmex/vmexflags.h>
#include <vmex/ioparams.h>
#include <vmex/vmexcnfg.h>
#include <vmex/pat.h>
#include <vmex/vmexerr.h>
#if (mtg_dummy)
        This is not a dummy driver !
#endif

#define NULL            0       /* null pointer */

uint lncintr();
uint lncopen();
uint lncclose();
extern uint t_create(), t_start(), t_restart(), t_suspend(), t_delete();
extern uint t_mode();
extern uint mm_l2p();
extern uint rn_getseg();
extern uint q_send();
extern uint ev_send(), ev_receive();
extern void k_fatal();
extern uint getditem(), claimvct(), activate();
extern uint blkcopy(), blkzero();
extern int printf();
extern void vperror();
#define Pr      (void) printf
extern void swab();             /* from the standard C library  */


/*****************************************************************************
* constants
*****************************************************************************/
#define LANVECT         0x44            /* LANCE interrupt vector       */
#define PCC_licr        V147_PCC->licr  /* LANCE int & contr reg.       */
#define LNC_rdp V147_LNC->le_rdp        /* LANCE Register Data Port     */
#define LNC_rap V147_LNC->le_rap        /* LANCE Register Address Port  */
#define ETH_addr        V147_NVRAM->nv_net.net_esd.station_addr
#define ETH_mask        V147_NVRAM->nv_net.net_esd.group_mask

#define LNCMASK         (ilev << 8)     /* mask for LANCE interrupts    */
#define STNAME          NAME("Lsrv")    /* name of the server task      */
#define STPRI           0x10            /* priority of the server task  */
#define STSTCK          0x800           /* sz of task supervisor stack  */
#define ST_GO           NAME("vai!")    /* command to start task        */
#define ST_HARAKIRI     NAME("mori")    /* command to restart (delete)  */
#define ST_MISS         0x00000001      /* missed packet event          */

#define M_LANCE147      0xccc0000       /* driver code for fatal error  */
#define F_MISALIGN      0x1101          /* wrong alloc.of LANCE struct. */
#define F_CORNVRAM      0x1102          /* NVRAM is corrupted           */
#define F_LNCINERR      0x1103          /* LANCE init error             */
#define F_TMODEERR      0x1104          /* error on t_mode()            */
```



```
#define F_L2PERROR      0x1105          /* error on mm_l2p()            */
#define F_TCREAERR      0x1106          /* error on t_create()          */
#define F_TSTRTERR      0x1107          /* error on t_start()           */
#define F_TRSTRERR      0x1108          /* error on t_restart()         */
#define F_UNEXIDON      0x1200          /* unexpected IDON interrupt    */
#define F_UNEXMERR      0x1201          /* memory error                 */
#define F_UNEXCERR      0x1202          /* heartbeat test failed        */
#define F_ISREVSND      0x1203          /* ISR error on ev_send()       */
#define F_ISRRCHKF      0x1204          /* ISR Rx consist. check failed */
#define F_ISRTCHKF      0x1205          /* ISR Tx consist. check failed */
#define F_ISRRSEND      0x1300          /* ISR Rx error on q_send()     */
#define F_ISRTSEND      0x1400          /* ISR Tx error on q_send()     */
        /* the last two hex digits are set to the error given by q_send */
#define F_READCHKF      0x1500          /* read() consist. check failed */
#define F_WRITCHKF      0x1501          /* writ() consist. check failed */


/*****************************************************************************
* structures and related constants
*****************************************************************************/

/*
 * command block
 */
typedef struct cbuf {
        struct cbuf     *c_nxt;         /* forward pointer              */
        struct le_md    *c_md;          /* pointer to message dsc       */
        caddr_t         c_buf;          /* pointer to buffer            */
        uint            c_qid;          /* queue identifier             */
        struct io_rw    *c_pargp;       /* pointer to parameter block   */
} cbuf_t;

/*
 * The big buffer (all uninitialized data is here)
 */
struct bigbuf {
        struct le_md    *bbrdr,         /* Receive Descriptor Ring      */
                        *bbtdr;         /* Transmit Descriptor Ring     */
        struct le_init_block bbiblk;    /* LANCE initialisation block   */
        cbuf_t          *bbrbf,         /* receive blocks               */
                        *bbtbf;         /* transmit blocks              */
        uint            bbivec,         /* interrupt vector number      */
                        bbilev,         /* interrupt level              */
                        bbdrlen,        /* dsc rings length (2's exp)   */
                        bbmdno,         /* number of md in each ring    */
                        bborcbndx,      /* oldest rcbuf index           */
                        bbotcbndx;      /* oldest tcbuf index           */
        int             bbbsyrcbno,     /* number of busy rcbuf's       */
                        bbbsytcbno;     /* number of busy tcbuf's       */
        struct {
                uint    st_tot,         /* total number of packets      */
                        st_def,         /* deferred packets count       */
                        st_one,         /* exactly one retry count      */
                        st_more,        /* more than one retry count    */
                        st_rtry,        /* retry errors                 */
                        st_miss;        /* missed packets               */
        } bbstat;                       /* structure for Ethernet tx statistics */
        int             bbwork;         /* flag for LANCE initialised   */
        uint            bbstid;         /* server task identifier       */
        long            bbstargp[4];    /* server task arguments        */
        char            bbwrka[18];     /* needed by VMEexec interface  */
};
#define work_area       (bigbuf->bbwrka)
```

```
#define ivec            (bigbuf->bbivec)
#define ilev            (bigbuf->bbilev)
#define drlen           (bigbuf->bbdrlen)
#define mdno            (bigbuf->bbmdno)
#define le_iblk         (bigbuf->bbiblk)
#define rring           (bigbuf->bbrdr)
#define tring           (bigbuf->bbtdr)
#define rcbuf           (bigbuf->bbrbf)
#define tcbuf           (bigbuf->bbtbf)
#define oldrcbndx       (bigbuf->bborcbndx)
#define oldtcbndx       (bigbuf->bbotcbndx)
#define oldrcb          (rcbuf[oldrcbndx])
#define oldtcb          (tcbuf[oldtcbndx])
#define busyrcb         (bigbuf->bbbsyrcbno)
#define busytcb         (bigbuf->bbbsytcbno)
#define currcb          (rcbuf[(oldrcbndx + busyrcb - 1) & (mdno - 1)])
#define curtcb          (tcbuf[(oldtcbndx + busytcb - 1) & (mdno - 1)])
#define working         (bigbuf->bbwork)
#define stat            (bigbuf->bbstat)
#define stid            (bigbuf->bbstid)
#define stargp          (bigbuf->bbstargp)


/*****************************************************************************
* lint and debug stuff
*****************************************************************************/
#ifdef lint
/* LINTLIBRARY */
uint t_create(nm,pr,sst,ust,flags,tid) uint nm,pr,sst,ust,flags,*tid;{return 0;}
uint t_start(tid,mod,adr,arg) uint tid,mod; void(*adr)(); long *arg; {return 0;}
uint t_restart(tid,arg) uint tid; long *arg; {return 0;}
uint t_suspend(tid) uint tid; {return 0;}
uint t_delete(tid) uint tid; {return 0;}
uint t_mode(mask,mode,oldmode) uint mask,mode,*oldmode; {return 0;}
uint mm_l2p(tid,lad,pad,len,mod) uint tid,*len,*mod;caddr_t lad,*pad;{return 0;}
uint rn_getseg(rnid,sz,flags,tmo,sp) uint rnid,sz,flags,tmo,*sp; {return 0;}
uint q_send(qid,msg) uint qid; struct ioc_cmsg *msg; {return 0;}
uint ev_send(tid,ev) uint tid,ev; {return 0;}
uint ev_receive(ev,flags,tm,evp) uint ev,flags,tm,*evp; {return 0;}
void k_fatal(err,flags) uint err,flags; {}
uint getditem(dev,off,item) uint dev,off,*item; {return 0;}
uint claimvct(vno,ha,arg,wa) uint vno,(*ha)(),arg; caddr_t wa; {return 0;}
uint activate(act,ms,opt,lev,id,arg) uint (*act)(),ms,opt,lev,id,arg;{return 0;}
uint blkcopy(from,to,sz,flags) caddr_t from,to; uint sz,flags; {return 0;}
uint blkzero(from,sz,flags) caddr_t from; uint sz,flags; {return 0;}
/*VARARGS1*/printf(s) char *s; {return 0;}
void vperror(s,err) char *s; uint err; {}
void swab(f,t,n) caddr_t f,t; int n; {}
#endif
```



```
/*
**      LANCE driver for VMEexec environment
**
**      %Z%%Y%%M%       %I%     %E%
*/

#ident  "%Z%%Y%%M%    %I%     %E%"

#include "lance.h"

#define lowtwobytes(a)  ((ushort)(a))
#define thirdbyte(a)    ((unchar)((uint)(a) >> 16))

#if task
/*
 * Server task
 */
void
Lserver(command, bigbuf)
uint                    command;
register struct bigbuf  *bigbuf;
{
        register uint   err;
        uint            events;

        /* if task restarted, suicide   */
        if (command == ST_HARAKIRI) {
            Pr("LANCE driver:reset\n");
            (void) t_delete(0);
            /* NOTREACHED */
        } else if (command != ST_GO) {
            Pr("LANCE driver:error on command for server task\n");
            (void) t_delete(0);
        }

        /* I am alive   */
        Pr("LANCE driver:initialised\n");

        /* wait for events forever      */
        while (!(err = ev_receive(ST_MISS, ANY, 0, &events))) {
            int oldpri;

            (void) t_setpri(0, 0xf0, &oldpri);
            Pr("LANCE driver:total %d missed packets till now\n", stat.st_miss);
            (void) t_setpri(0, oldpri, &oldpri);
        }

        /* error        */
        vperror("LANCE driver: ev_receive():", err);
        (void) t_delete(0);

        /* NOTREACHED */
} /* Lserver() */
#endif


/*
 * Stops LANCE activity
 */
static void
lstop()
{
        PCC_licr = 0;   /* disable interrupts   */
```

```
        LNC_rap = LE_CSR0;      /*    stop      */
        LNC_rdp = LE_STOP;      /*  the LANCE   */
} /* lstop */



uint
lncinit(dev, tid, pargp, rval)
uint            dev, tid;
caddr_t         *rval;
struct io_init  *pargp;
{
        uint            drvspace_sz, tmpmdno, tmpdrlen, drvspace;
        struct bigbuf   *bigbuf;

        /* initialise nvram Ethernet address    */
        if ((V147_NVRAM->nv_bug.bug_eadr[0] & 0xf0) != 0x20)
                /* if test fails, nvram has been corrupted      */
                k_fatal(C_DRIVER | M_LANCE147 | F_CORNVRAM, FATAL_FLAG);
        ETH_addr[0] = 0x08;
        ETH_addr[1] = 0x00;
        ETH_addr[2] = 0x3e;
        (void) blkcopy((caddr_t)V147_NVRAM->nv_bug.bug_eadr,
                       (caddr_t)&ETH_addr[3], 3, LP_ADDR);

        /* establish the size of the rings      */
        if (getditem(dev, D_ACMDS, &tmpmdno))
                k_fatal(C_DRIVER | M_LANCE147 | F_GETDITEM, FATAL_FLAG);
        tmpmdno++; tmpmdno /= 2;
        for (tmpdrlen = 0; tmpdrlen < 8; tmpdrlen++)
                if (tmpmdno <= (1<<tmpdrlen))
                        break;
        tmpmdno = 1<<tmpdrlen;

        /* get uninitialised memory from region 0 and zero it   */
        drvspace_sz = 8 + 2*tmpmdno*(sizeof(struct le_md) + sizeof(cbuf_t)) +
                      sizeof(struct bigbuf);
        if (rn_getseg(0, drvspace_sz, NOWAIT, 0, &drvspace))
                k_fatal(C_DRIVER | M_LANCE147 | F_NOMEM, FATAL_FLAG);
        (void) blkzero((caddr_t)drvspace, drvspace_sz, LP_ADDR);

        /* set bigbuf, drlen, mdno, rring, tring, rcbuf, tcbuf  */
        /* and the first argument of the server task            */
        bigbuf = (struct bigbuf *)
          (drvspace + 8 + 2*tmpmdno*(sizeof(struct le_md) + sizeof(cbuf_t)));
        drlen  = tmpdrlen;
        mdno   = tmpmdno;
        rring  = (struct le_md *) ((drvspace + 0x7) & 0xfffffff8);
        tring  = &rring[mdno];
        rcbuf  = (cbuf_t *)(&tring[mdno]);
        tcbuf  = &rcbuf[mdno];

        /* check the addresses of le_iblk, rring and tring      */
        if (((uint)rring & 0xff000007) || ((uint)tring & 0xff000007) ||
            ((uint)(&le_iblk) & 0xff000001))
                k_fatal(C_DRIVER | M_LANCE147 | F_MISALIGN, FATAL_FLAG);

        /* get vmexgen interrupt level and vector number        */
        if (getditem(dev, D_IVEC, &ivec))
                k_fatal(C_DRIVER | M_LANCE147 | F_GETDITEM, FATAL_FLAG);
        if (getditem(dev, D_ILEV, &ilev))
                k_fatal(C_DRIVER | M_LANCE147 | F_GETDITEM, FATAL_FLAG);
```



```
        /* link interrupt routine to vector     */
        (void) claimvct(ivec, lncintr, bigbuf, work_area);

        *rval = (caddr_t)bigbuf;
        return DE_NOERR;
} /* lncinit */



uint lncopen(dev, tid, pargp, bigbuf, rval)
uint                    dev, tid, *rval;
struct io_oc            *pargp;
register struct bigbuf  *bigbuf;
{
        register int    i;
        uint            oldmode;
        int             retval = DE_NOERR;

        /* enter critical section       */
        if (t_mode(NOPREEMPT | LEVEL, NOPREEMPT | LNCMASK, &oldmode))
                k_fatal(C_DRIVER | M_LANCE147 | F_TMODEERR, FATAL_FLAG);

        /* verify that driver is not initialised        */
        if (working)
                { retval = DE_ICMD; goto end; }
        working = 1;

        /* stop the LANCE       */
        lstop();
#if task
        /* create and start the server task     */
        if (t_create(STNAME, STPRI, STSTCK, 0, 0, &stid))
                k_fatal(C_DRIVER | M_LANCE147 | F_TCREAERR, FATAL_FLAG);
        stargp[0] = ST_GO;
        stargp[1] = (long)bigbuf;
        if (t_start(stid, SUPV, Lserver, stargp))
                k_fatal(C_DRIVER | M_LANCE147 | F_TSTRTERR, FATAL_FLAG);
#endif
        /* set the address of the initialisation block  */
        LNC_rap = LE_CSR1;
        LNC_rdp = lowtwobytes(&le_iblk);
        LNC_rap = LE_CSR2;
        LNC_rdp = thirdbyte(&le_iblk);

        /* set the LANCE hardware parametres    */
        LNC_rap = LE_CSR3;
        LNC_rdp = LE_BSWP | LE_BCON;

        /* reset the Register Address Port to CSR0      */
        LNC_rap = LE_CSR0;

        /* fill the initialisation block        */
        le_iblk.ib_prom = 0;            le_iblk.ib_intl = 0;
        le_iblk.ib_drty = 0;            le_iblk.ib_coll = 0;
        le_iblk.ib_dtcr = 0;            le_iblk.ib_loop = 0;
        le_iblk.ib_dtx  = 0;            le_iblk.ib_drx  = 0;

        swab((caddr_t)ETH_addr, (caddr_t)le_iblk.ib_padr, 6);
        swab((caddr_t)ETH_mask, (caddr_t)le_iblk.ib_ladrf, 8);

        le_iblk.ib_rdrp.drp_len  = drlen;
        le_iblk.ib_rdrp.drp_drah = thirdbyte(rring);
        le_iblk.ib_rdrp.drp_dral = lowtwobytes(rring);
```

```
        le_iblk.ib_tdrp.drp_len  = drlen;
        le_iblk.ib_tdrp.drp_drah = thirdbyte(tring);
        le_iblk.ib_tdrp.drp_dral = lowtwobytes(tring);

        /* initialise the LANCE */
        LNC_rdp = LE_INIT;

        /* set rcbuf and tcbuf indexes  */
        oldrcbndx = oldtcbndx = 0;
        busyrcb = busytcb = 0;

        /* set up the receive and transmit blocks chains        */
        for (i = 0; i < mdno; i++) {
                rcbuf[i].c_md  = &rring[i];
                rcbuf[i].c_nxt = &rcbuf[i+1];
                tcbuf[i].c_md  = &tring[i];
                tcbuf[i].c_nxt = &tcbuf[i+1];
        }
        rcbuf[i-1].c_nxt = rcbuf;
        tcbuf[i-1].c_nxt = tcbuf;

        /* initialise the receive and transmit rings    */
        for (i = 0; i < mdno; i++) {
                rring[i].md_own  = 0;   rring[i].md_err  = 0;
                rring[i].md_fram = 0;   rring[i].md_oflo = 0;
                rring[i].md_crc  = 0;   rring[i].md_buff = 0;
                rring[i].md_stp  = 0;   rring[i].md_enp  = 0;
                rring[i].md_mcnt = 0;   rring[i].md_bcnt = -MAXRBUF;
        }

        for (i = 0; i < mdno; i++) {
                tring[i].md_own  = 0;   tring[i].md_err    = 0;
                tring[i].md_more = 0;   tring[i].md_one    = 0;
                tring[i].md_def  = 0;   tring[i].md_stp    = 1;
                tring[i].md_enp  = 1;   tring[i].md_errflg = 0;
        }

        /* reset the stats counters     */
        (void) blkzero(&stat, sizeof(stat), LP_ADDR);

        /* wait for initialization done with timeout    */
        if (activate((uint)k_fatal, 20, 0, 7,
            C_DRIVER | M_LANCE147 | F_LNCINERR, FATAL_FLAG))
                k_fatal(C_DRIVER | M_LANCE147 | F_ACTIVATE, FATAL_FLAG);
        while(!(LNC_rdp & LE_INTR))
                ;
        if (activate((uint)k_fatal, 20, CANCEL, 7,
            C_DRIVER | M_LANCE147 | F_LNCINERR, FATAL_FLAG))
                k_fatal(C_DRIVER | M_LANCE147 | F_ACTIVATE, FATAL_FLAG);
        if (LNC_rdp != (LE_IDON | LE_INTR | LE_INIT))
                k_fatal(C_DRIVER | M_LANCE147 | F_LNCINERR, FATAL_FLAG);
        LNC_rdp = LE_IDON;

        /* start the LANCE      */
        if      (ilev < PCC_IL1)        ilev = PCC_IL1;
        else if (ilev > PCC_IL6)        ilev = PCC_IL6;
        PCC_licr = PCC_IEN | ilev;
        LNC_rdp = LE_INEA | LE_STRT;

end:    /* exit critical section        */
        if (t_mode(NOPREEMPT | LEVEL, oldmode, &oldmode))
                k_fatal(C_DRIVER | M_LANCE147 | F_TMODEERR, FATAL_FLAG);
```



```
        *rval = 0; return retval;
} /* lncopen */



uint lncclose(dev, tid, pargp, bigbuf, rval)
uint                    dev, tid, *rval;
struct io_oc            *pargp;
register struct bigbuf  *bigbuf;
{
        uint    oldmode;
        int     retval = DE_NOERR;

        /* enter critical section       */
        if (t_mode(NOPREEMPT | LEVEL, NOPREEMPT | LNCMASK, &oldmode))
                k_fatal(C_DRIVER | M_LANCE147 | F_TMODEERR, FATAL_FLAG);

        /* verify that driver has been initialised      */
        if (!working)
                { retval = DE_DOWN; goto end; }
        working = 0;

        /* stop the LANCE       */
        lstop();
#if task
        /* restart the server task      */
        stargp[0] = ST_HARAKIRI;
        stargp[1] = (long)bigbuf;
        if (t_restart(stid, stargp))
                k_fatal(C_DRIVER | M_LANCE147 | F_TRSTRERR, FATAL_FLAG);
#endif
end:    /* exit critical section        */
        if (t_mode(NOPREEMPT | LEVEL, oldmode, &oldmode))
                k_fatal(C_DRIVER | M_LANCE147 | F_TMODEERR, FATAL_FLAG);
        *rval = 0; return retval;
} /* lncclose */



uint lncread(dev, tid, pargp, bigbuf, rval)
uint                    dev, tid, *rval;
struct io_rw            *pargp;
register struct bigbuf  *bigbuf;
{
        uint    oldmode, len, mod;
        int     retval = DE_NOERR;

        /* enter critical section       */
        if (t_mode(NOPREEMPT | LEVEL, NOPREEMPT | LNCMASK, &oldmode))
                k_fatal(C_DRIVER | M_LANCE147 | F_TMODEERR, FATAL_FLAG);

        /* verify that the driver has been initialised  */
        if (!working)
                { retval = DE_DOWN; goto end; }

        /* too many commands ?  */
        if (busyrcb == mdno)
                { retval = DE_BUSY; goto end; }

        /* check command parametres     */
        if (!(pargp->io_flags & ASYNC) || !(pargp->io_flags & FQ_NOBLK) ||
            pargp->io_cnt != MAXRBUF)
                { retval = DE_ICMD; goto end; }
```

```
        /* fill a receive command block */
        if (++busyrcb <= 0 || busyrcb > mdno) {
                k_fatal(C_DRIVER | M_LANCE147 | F_READCHKF, FATAL_FLAG);
        }
        if (pargp->io_flags & LP_ADDR) {
                currcb.c_buf = pargp->io_bufp;
        } else {
                if (mm_l2p(tid, pargp->io_bufp, &currcb.c_buf, &len, &mod))
                        k_fatal(C_DRIVER | M_LANCE147 | F_L2PERROR, FATAL_FLAG);
                if ((len < MAXRBUF) || (mod & WPROTECT) || !(mod & NODCACHE))
                        { --busyrcb; retval = DE_ICMD; goto end; }
        }
        currcb.c_qid = pargp->io_qid;
        currcb.c_pargp = pargp;
        currcb.c_md->md_hadr = thirdbyte(currcb.c_buf);
        currcb.c_md->md_ladr = lowtwobytes(currcb.c_buf);
        currcb.c_md->md_own  = 1;

end:    /* exit critical section        */
        if (t_mode(NOPREEMPT | LEVEL, oldmode, &oldmode))
                k_fatal(C_DRIVER | M_LANCE147 | F_TMODEERR, FATAL_FLAG);
        *rval = 0; return retval;
} /* lncread */



uint lncwrite(dev, tid, pargp, bigbuf, rval)
uint                    dev, tid, *rval;
struct io_rw            *pargp;
register struct bigbuf  *bigbuf;
{
        uint    oldmode, len, mod;
        int     retval = DE_NOERR;

        /* enter critical section       */
        if (t_mode(NOPREEMPT | LEVEL, NOPREEMPT | LNCMASK, &oldmode))
                k_fatal(C_DRIVER | M_LANCE147 | F_TMODEERR, FATAL_FLAG);

        /* verify that the driver has been initialised  */
        if (!working)
                { retval = DE_DOWN; goto end; }

        /* too many commands ?  */
        if (busytcb == mdno)
                { retval = DE_BUSY; goto end; }

        /* check command parametres     */
        if (!(pargp->io_flags & ASYNC) || !(pargp->io_flags & FQ_NOBLK) ||
            pargp->io_cnt < ETH_MIN_TU || pargp->io_cnt > ETH_MAX_TU)
                { retval = DE_ICMD; goto end; }

        /* fill a transmit command block        */
        if (++busytcb <= 0 || busytcb > mdno) {
                k_fatal(C_DRIVER | M_LANCE147 | F_WRITCHKF, FATAL_FLAG);
        }
        curtcb.c_md->md_bcnt = -pargp->io_cnt;
        if (pargp->io_flags & LP_ADDR) {
                curtcb.c_buf = pargp->io_bufp;
        } else {
                if (mm_l2p(tid, pargp->io_bufp, &curtcb.c_buf, &len, &mod))
                        k_fatal(C_DRIVER | M_LANCE147 | F_L2PERROR, FATAL_FLAG);
                if (len < pargp->io_cnt)
```



```
                        { --busytcb; retval = DE_NOMEM; goto end; }
        }
        curtcb.c_qid = pargp->io_qid;
        curtcb.c_pargp = pargp;
        curtcb.c_md->md_hadr = thirdbyte(curtcb.c_buf);
        curtcb.c_md->md_ladr = lowtwobytes(curtcb.c_buf);
        curtcb.c_md->md_own  = 1;

        /* hasten LANCE response        */
        LNC_rdp = LE_TDMD | LE_INEA;

end:    /* exit critical section        */
        if (t_mode(NOPREEMPT | LEVEL, oldmode, &oldmode))
                k_fatal(C_DRIVER | M_LANCE147 | F_TMODEERR, FATAL_FLAG);
        *rval = 0; return retval;
} /* lncwrite */



uint lnccntrl() {return DE_ICMD;}



uint
lncintr(bigbuf)
struct bigbuf   *bigbuf;
{
        register ushort csr0 = LNC_rdp;
        register uint   err;
        struct ioc_cmsg msg;

        /* check for fatal errors       */
        if (csr0 & (LE_CERR | LE_MERR | LE_IDON)) {
            LNC_rdp = LE_STOP;
            working = 0;
            if (csr0 & LE_IDON)
                k_fatal(C_DRIVER | M_LANCE147 | F_UNEXIDON, FATAL_FLAG);
            if (csr0 & LE_MERR)
                k_fatal(C_DRIVER | M_LANCE147 | F_UNEXMERR, FATAL_FLAG);
            if (csr0 & LE_CERR)
                k_fatal(C_DRIVER | M_LANCE147 | F_UNEXCERR, FATAL_FLAG);
            /* NOTREACHED */
        }

        /* missed packet ?      */
        if (csr0 & LE_MISS) {
            ++stat.st_miss;
#if task
            if (ev_send(stid, ST_MISS))
                k_fatal(C_DRIVER | M_LANCE147 | F_ISREVSND, FATAL_FLAG);
#endif
        }

        /* Rx interrupt ?       */
        if (csr0 & LE_RINT)
            while (busyrcb && !oldrcb.c_md->md_own) {

                /* tell the task */
                msg.ioc_csr1 = (uint)oldrcb.c_pargp;
                if (oldrcb.c_md->md_err ||
                    !oldrcb.c_md->md_stp || !oldrcb.c_md->md_enp) { /* errors */
                        msg.ioc_mt = IOCMT_C;   /* completion   */
                        msg.ioc_ss = 1;         /* driver error */
```

```
                msg.ioc_eeee = ((ushort *)(oldrcb.c_md))[1];
        } else {                        /* no errors    */
                msg.ioc_error = 0;
        }
        msg.ioc_csr2 = oldrcb.c_md->md_mcnt - 4;
        if (err = q_send(oldrcb.c_qid, &msg)) {
                LNC_rdp = LE_STOP;
                k_fatal(C_DRIVER | M_LANCE147 | F_ISRRSEND | err,
                        FATAL_FLAG);
        }

        /* zero the receive count       */
        oldrcb.c_md->md_mcnt = 0;

        /* update the indexes and make consistency check        */
        if (--busyrcb < 0 || busyrcb > mdno-1)
                k_fatal(C_DRIVER | M_LANCE147 | F_ISRRCHKF, FATAL_FLAG);
        oldrcbndx = (oldrcbndx + 1) & (mdno - 1);

    } /* while !md_own && busyrcb */
/* Rx interrupt */

/* Tx interrupt (or babbling transmitter) ?     */
if (csr0 & (LE_BABL | LE_TINT))
    while (busytcb && !oldtcb.c_md->md_own) {

        /* gather statistics    */
        if (oldtcb.c_md->md_def)
                { oldtcb.c_md->md_def = 0; ++stat.st_def; }
        if (oldtcb.c_md->md_one)
                { oldtcb.c_md->md_one = 0; ++stat.st_one; }
        else if (oldtcb.c_md->md_more)
                { oldtcb.c_md->md_more = 0; ++stat.st_more; }
        ++stat.st_tot;

        /* tell the task */
        msg.ioc_csr1 = (uint)oldtcb.c_pargp;
        if (oldtcb.c_md->md_errflg || (csr0 & LE_BABL)) { /* errors */
                msg.ioc_mt = IOCMT_C;   /* completion               */
                msg.ioc_ss = 1;         /* driver error             */
                if (oldtcb.c_md->md_errflg)
                        msg.ioc_eeee = oldtcb.c_md->md_errflg;
                else
                        msg.ioc_eeee = 0xbab1;  /* babble error     */
        } else {                        /* no errors    */
                msg.ioc_error = 0;
        }
        msg.ioc_csr2 = -oldtcb.c_md->md_bcnt;
        if (err = q_send(oldtcb.c_qid, &msg)) {
                LNC_rdp = LE_STOP;
                k_fatal(C_DRIVER | M_LANCE147 | F_ISRTSEND | err,
                        FATAL_FLAG);
        }

        /* update the indexes   */
        if (--busytcb < 0 || busytcb > mdno-1)
                k_fatal(C_DRIVER | M_LANCE147 | F_ISRTCHKF, FATAL_FLAG);
        oldtcbndx = (oldtcbndx + 1) & (mdno - 1);
    } /* while md_own && busytcb */
/* Tx interrupt */

/* reset all interrupt conditions       */
LNC_rdp = csr0;
```



```
        return 0;
} /* lncintr() */
```